

Г. МАЙЕРС

АРХИТЕКТУРА  
СОВРЕМЕННЫХ

ЗДАНИЙ







# **АРХИТЕКТУРА СОВРЕМЕННЫХ ЭВМ**

# **ADVANCES IN COMPUTER ARCHITECTURE**

**SECOND EDITION**

---

**GLENFORD J. MYERS**

**INTEL CORPORATION  
SANTA CLARA, CALIFORNIA**

**A WILEY-INTERSCIENCE PUBLICATION**  
**JOHN WILEY & SONS**  
**NEW YORK CHICHESTER BRISBANE TORONTO SINGAPORE**  
**1982**

Г. МАЙЕРС

# АРХИТЕКТУРА СОВРЕМЕННЫХ ЗДАНИЙ

2

В 2-Х КНИГАХ

---

ПЕРЕВОД С АНГЛИЙСКОГО  
ПОД РЕДАКЦИЕЙ КАНД. ТЕХН. НАУК  
В. К. ПОТОЦКОГО



МОСКВА «МИР» 1985

ББК 32.973

М 14

УДК 681.3

Майерс Г.

М 14 Архитектура современных ЭВМ: В 2-х кн.  
Кн. 2 — Пер. с англ. — М.: Мир, 1985. — 312 с., ил.

Монография известного американского специалиста в области вычислительной техники посвящена анализу фундаментальных проблем архитектуры современных вычислительных машин; описывается ряд достижений на пути преодоления этих проблем. Книга 2 содержит критический анализ достоинств и недостатков архитектуры ряда современных вычислительных систем нетрадиционной архитектуры (SWARD, IAPX 432), машин базы данных (RAP, CASSM, DBC) и машин потоков данных. Кроме того, здесь исследуются проблемы оптимизация архитектуры вычислительных систем.

Для специалистов в области вычислительной техники и программирования. Может быть полезна студентам старших курсов соответствующих специальностей вузов.

2405000000-243  
М 041(01)-85 165-85, ч. 1

ББК 32.973

6Ф7.3

Редакция литературы по информатике и электронике

Copyright © 1978 by John Wiley & Sons, Inc. All rights reserved. Authorized translation from English language edition published by John Wiley & Sons, Inc.

© Перевод на русский язык, «Мир», 1985

---

## ГЛАВА 15

### НАБОР КОМАНД СИСТЕМЫ SWARD

В этой главе рассматривается основной набор команд системы SWARD. Для большинства команд справедливы следующие общие замечания.

1. Если операндами команд являются два массива, они должны быть конформными друг другу, т. е. иметь одинаковую размерность и равное количество элементов по соответствующим размерностям. Это же утверждение относится и к срезам массивов. Во всех случаях, когда указывается, что операндом команды является массив, если отсутствует специальная оговорка, операндом может быть и срез массива (множество значений однотипных компонентов записей, являющихся элементами массива).

2. Если операндами команд являются две записи, у них должны быть одинаковые атрибуты, т. е. записи должны содержать одинаковое количество элементов и соответствующие компоненты должны иметь одинаковые атрибуты (теги).

3. Если операндом команды может быть ячейка определенного типа, в качестве операнда может использоваться и ячейка с вложенным тегом (тегами)<sup>1)</sup>. Например, если операнд должен представлять собой целое число, адрес операнда может быть адресом ячейки, содержащей целое число, элемент массива целых чисел, информацию о косвенном доступе к целочисленным данным, целочисленный параметр, целочисленный компонент записи, целочисленные данные, тип которых задается пользователем, и т. д.

4. При выполнении многих команд могут регистрироваться ошибки, принадлежащие некоторому основному набору ошибок. В упомянутый набор ошибок входят ошибки следующих типов:

---

<sup>1)</sup> Вложенный тег должен совпадать с тегом для типа ячейки, указанного в качестве допустимого для операнда. Это и иллюстрируется следующим примером. — Прим. перев.

неправильная адресация, данные неизвестного формата, нарушение защиты доступа, недействительный потенциальный адрес, выход индекса за допустимые границы, неверный тип операнда, неопределенный операнд, несовместимые операнды, имитация ошибки и неопределенный доступ к операнду.

5. Под операндом арифметического типа понимают любой из следующих операндов: целое число, целое число увеличенной разрядности, порядковое число, десятичное число с фиксированной точкой, десятичное число с плавающей точкой или литерал. Под операндом «строка» (или «поле») подразумевается строка (или поле), состоящая из символов или токенов. Под операндом символьного типа понимают поле или строку символов. Операнд логического типа обозначает либо порядковые числа, имеющие значения 0 и 1, либо литералы с такими же значениями.

6. В описаниях форматов команд первое поле содержит код операции. Длина этого поля составляет 1—5 токен в зависимости от конкретной команды. Для адреса операнда используется мнемоническое обозначение ОА, для адреса команды — IA.

7. В качестве адресов операндов допускается использование литералов, кроме тех случаев, когда при выполнении команды значение операнда может быть изменено или когда операнд не может быть арифметическим (числовым) или логическим.

8. Длиной поля символов или токенов считается значение в поле «размер» тега ячейки, длиной строки символов или токенов — значение в поле «длина» текущего содержимого ячейки.

9. Применительно к системе SWARD представления о командах как о «квантах» обработки не являются правомочными<sup>1)</sup>. В частности, ошибки некоторых типов могут повлечь получение частичных результатов, хотя никогда не смогут внести дезорганизацию на уровне основных объектов системы (например, проявиться в «потере» отдельных частей памяти или объектов). Кроме того, если две или более процесс-машины одновременно выполняют команды, адресующиеся к одним и тем же компонентам системы (например, к ячейке объекта «память данных»), то система не гарантирует ни порядка их выполнения, ни того, что при их реализации не произойдет непредвиденный интерференционный эффект.

10. Если при выполнении команды возможно появление нескольких ошибок, то имеет место и регистрируется системой только одна из них без указания ее очередности (т. е. архитек-

<sup>1)</sup> Иными словами, нельзя делать утверждение относительно команд, что они либо выполняются полностью, либо совсем не выполняются. — Прим. перев.

турой системы SWARD не предусмотрено какое-либо упорядочение регистрации одновременно возникающих ошибок).

11. В целях повышения практической эффективности работы системы при реализации микропрограмм для команд сравнения EQBF, NEBF, LTBF, LEBF, GTBF, GEBF и SEARCH допускается небольшое отклонение от общего правила выявления неопределенных значений. В том и только в том случае, когда операнд является полем или записью (или массивом полей или записей), поле или запись, которые частично не определены, не вызовут появления ошибки «неопределенный операнд». В конкретной реализации системы такие операнды могут использоваться для команд сравнения, выполнение которых завершается признаком результата «не сравниваются» (неравенство операндов). Однако, если поле или запись полностью не определены, при выполнении команды регистрируется ошибка.

## КОМАНДЫ ОБЩЕГО НАЗНАЧЕНИЯ

### *Имя команды. MOVE*

*Выполняемая операция.* Пересылка второго операнда (копирование) на место первого операнда.

*Формат.* 1, OA, OA

*Операнды*<sup>1)</sup>. Оба операнда должны быть взаимно совместимыми, т. е. должны быть числами, полями или строками символов или токенов, указателями или записями. Оба операнда могут быть одновременно массивами (или их срезами); при этом предполагается пересылка значений между соответствующими элементами этих массивов. Возможна и такая ситуация, когда массивом является только первый операнд; тогда выполнение команды сводится к записи значений второго операнда на место каждого элемента первого операнда.

Если оба операнда представляют собой числа разного типа или размера, то результат, размещаемый на месте первого операнда, преобразуется в форму, соответствующую этому операнду. При выполнении команды MOVE округления не производится. Когда выполняется пересылка строки или поля в строку, длина первого операнда устанавливается равной длине второго операнда. Если пересылка производится в поле символов и второй операнд короче первого, то остальная (правая) часть первого операнда заполняется пробелами. Если же пересылка

<sup>1)</sup> В этом разделе описывается назначение всех полей адреса каждой рассматриваемой команды, а не только их подмножества, отвечающего введенному автором определению операнда (см. раздел «Форматы команд и способы адресации» в гл. 14), которого автор, за незначительными исключениями (сравните нумерацию операндов в командах X MACHINE и TRACE), придерживается и в этих описаниях. — Прим. перев.

осуществляется в поле токенов и второй операнд короче первого, избыточная (левая) часть первого операнда заполняется нулями. Допускаются любые сочетания операндов полей символов, полей токенов, строк символов и строк токенов. При пересылке символов в поле или строку токенов или наоборот выполняется побитовое копирование (т. е. какое-либо преобразование информации отсутствует, за исключением возможного изменения атрибута длины и заполнения соответствующими символами избыточных полей).

Если одна запись пересыпается в другую, у них должно быть одинаковое число компонентов и соответствующие компоненты должны иметь одинаковые атрибуты. Пересылка записи эквивалентна пересылке всех ее отдельных компонентов. *Ошибки.* Основной набор ошибок (за исключением «неверный тип операнда») и «переполнение».

### Имя команды. CONVERT

*Выполняемая операция.* Пересылка второго операнда на место первого операнда. Если операнды различаются по типу, то выполняется одно из предусмотренных преобразований.

*Формат.* 09, ОА, ОА

*Операторы.* Кроме специально оговариваемых случаев, действуют те же правила, что и при выполнении команды MOVE. Однако требования на совместимость операндов менее жесткие. В табл. 15.1 представлены все правила выполняемых преобра-

Таблица 15.1. Правила преобразования

Тип операнда 2											
		ц	ц у р	пч	д ф т	д п т	п с	п т	с с	с т	
Тип операнда 1	ц	1	1	1	1	1	3	2	3	2	
	цур	1	1	1	1	1	3	2	3	2	
	пч	1	1	1	1	1	3	2	3	2	
	дфт	1	1	1	1	1	3		3		
	дпт	1	1	1	1	1	3		3		
	пс	4	4	4	5	6	1	7	1	7	
	пт	8	8	8			9	1	9	1	
	сс	4	4	4	5	6	1	7	1	7	
	ст	8	8	8			9	1	9	1	

*Обозначения.* ц — целое, цур — целое увеличенной разрядности, пч — порядковое число, дфт — десятичное с фиксированной точкой, дпт — десятичное с плавающей точкой, пс — поле символов, пт — поле токенов, сс — строка символов, ст — строка токенов; *<b>* — пробел, *<s>* — точка в десятичном представлении числа (сокращение обозначается символом *-*) или знак *<>*, *<t>* — знак *<>* или пробел, *<d>* — любая цифра в диапазоне 0—9, *<e>* — любая цифра в диапазоне 1—9, *<h>* — цифра в диапазоне 0—9 или буква от А до F, *<hb>* — любой элемент из *<h>* или *<b>*, квадратные скобки обозначают необязательный элемент, а ... — повторение предыдущего символа. В таблице представлены следующие правила преобразования:

1. Пересылка данных подобно той, которая выполняется по команде MOVE.

2. В первый операнд непосредственно пересыпаются последние 6, 12 или 2 token (соответственно для преобразования в целое, целое увеличенной разрядности или порядковое число). Если длина второго операнда больше 6, 12 или 2 token соответственно, то во всех остальных ведущих (левых) tokenах должны быть нули. Если второй операнд имеет длину меньше 6, 12 или 2 token соответственно, то он считается дополненным слева нулями до соответствующей длины.

3. Символьная величина должна иметь одну из следующих форм:

$$[<b>...][<s>][<b>...][<d>...][\cdot][<d>...][<b>...]$$

$$[<b>...][<s>][<b>...][<d>...][\cdot][<d>...]\text{E} [<s>] D...$$

4. Целое число преобразуется в одну из следующих форм:

$$<t> <e>[<d>...] \text{или } <b>0.$$

*Примечание.* Если первый операнд — строка символов, результат выравнивается по левому концу строки (подобно команде MOVE). Если первый операнд — поле символов, результат выравнивается по правому концу строки, а если занимает не всю строку, то слева дополняется пробелами.

5. Десятичное число с фиксированной точкой преобразуется в одну из следующих форм:

$$<m> <e>[<d>...][\cdot<d>...] \text{или } <b>0[\cdot<d>...]$$

См. примечание к правилу 4.

6. Десятичное число с плавающей точкой приводится к следующему виду:

$$<m>0.<e>[<d>...] \text{E} <s> <d>.$$

См. примечание к правилу 4.

Число цифр в дробной части равно содержимому поля «размер мантиссы» во втором операнде.

7. Содержимое tokenов воспринимается как символьная величина из множества  $<h>$ .

8. Целое число воспринимается как содержимое 6, 12 или 2 token соответственно.

9. Исходная величина должна состоять из символов множества  $<h>$ . Пробелы преобразуются в tokenы с нулевыми значениями.

зований. Пробел в таблице означает, что преобразование не выполняется и регистрируется ошибка «несовместимые операнды». Если при попытке выполнения преобразования значение второго операнда не удовлетворяет правилам преобразования, регистрируется ошибка «недопустимое преобразование данных». В качестве operandов не могут использоваться полностью массивы или записи.

*Ошибки.* Основной набор ошибок, а также ошибки типа «недопустимое преобразование данных» и «переполнение».

### Имя команды. DISPLAY-OPERAND-TYPE (DOT)

*Выполняемая операция.* Пересылка тега с атрибутами второго операнда в первый операнд.

*Формат.* 0008, OA, OA

*Операнды.* Первый operand должен быть полем или строкой tokenов. Второй operand может быть operandом любого типа, кроме литерала. Машинна определяет тег второго operand'a стандартным образом и помещает копию этого тега в первый operand. В частности, если адрес operand'a — это адрес ячейки «параметр» или «косвенный доступ к данным», в первый operand записывается значение вложенного тега. Если же адрес operand'a указывает на компонент записи или элемент массива, записи подлежит значение тега компонента или элемента, а не

тега записи или массива. Если адрес операнда — это адрес ячейки, тип которой определяется пользователем, то записывается тег этой ячейки.

Возникновение ошибки «переполнение» блокируется. Если длина тега превышает размер первого операнда, последний целиком заполняется токенами из тега, начиная с его начала. *Ошибки*. Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

*Примечания*. Хотя адрес второго операнда может быть адресом ячейки любого типа, в основном он используется для обращения к ячейкам «параметр» или «косвенный доступ к данным» с динамически определяемыми типом, границами или размером. Использование данной команды позволяет определить текущее значение атрибутов ячеек «параметр» или «косвенный доступ к данным».

#### **Имя команды. UNDEFINE (UNDEF)**

*Выполняемая операция*. Присвоение операнду признака неопределенности его значения.

*Формат*. 0001, ОА

*Операторы*. Допускаются операнды произвольного типа. Если операнд является полем токенов, то данная команда никаких операций не выполняет. Если операндом определяется группа ячеек (массив, срез массива или запись), неопределенное значение получает каждый элемент или компонент.

*Ошибки*. Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

### **АРИФМЕТИЧЕСКИЕ КОМАНДЫ**

#### **Имя команды. ADD**

*Выполняемая операция*. Сложение двух операндов и запись результата на место первого операнда.

*Формат*. 2, ОА, ОА

*Операторы*. Оба операнда должны быть арифметическими, т. е. числами. Оба операнда одновременно могут быть массивами; при этом выполняется их поэлементное сложение. Если первый операнд — массив, а второй — скалярная величина, значение второго операнда добавляется к значениям всех элементов первого.

Если операнды отличаются по типу или размеру, второй операнд преобразуется таким образом, чтобы возможно было выполнение операции сложения, после чего и выполняется эта операция. Результат в виде десятичного числа с плавающей точкой всегда нормализуется.

**Ошибки.** Основной набор ошибок, а также ошибки типа «переполнение» и «потеря значимости».

**Имя команды.** SUBTRACT (SUB)

**Выполняемая операция.** Вычитание второго операнда из первого и запись результата на место первого операнда.

**Формат.** 3, OA, OA

**Операнды.** См. описание команды ADD

**Ошибки.** Основной набор ошибок, а также ошибки типа «переполнение» и «потеря значимости».

**Имя команды.** MULTIPLY (MULT)

**Выполняемая операция.** Перемножение двух operandов и запись результата на место первого операнда.

**Формат.** 4, OA, OA

**Операнды.** См. описание команды ADD

**Ошибки.** Основной набор ошибок, а также ошибки типа «переполнение» и «потеря значимости».

**Примечание.** Если operandы являются массивами, то выполняется поэлементное перемножение, а не так называемое матричное перемножение.

**Имя команды.** DIVIDE

**Выполняемая операция.** Деление первого операнда на второй operand и запись результата — частного на место первого операнда. Если первый operand — целое число, то результат представляется в виде целого числа, величина которого имеет максимальное целочисленное значение, не превышающее абсолютной величины частного, а знак совпадает со знаком последнего.

**Формат.** 02, OA, OA

**Операнды.** См. описание команды ADD

**Ошибки.** Основной набор ошибок, а также ошибки типа «переполнение», «потеря значимости» и «некорректное деление».

**Имя команды.** REMAINDER

**Выполняемая операция.** Деление первого операнда на второй и запись остатка на место первого операнда.

**Формат.** 0002, OA, OA

**Операнды.** Operandы могут быть целыми числами, в том числе увеличенной разрядности, или порядковыми числами. Оба operandы могут быть либо массивами, либо скалярами или первый operand может быть массивом, а второй — скаляром.

Знак результата равен знаку первого operandа.

**Ошибки.** Основной набор ошибок, а также ошибки типа «переполнение» и «некорректное деление».

**Имя команды.** ABSOLUTE (ABS)

**Выполняемая операция.** Присвоение операнду положительного знака.

**Формат.** 01, ОА

**Операнды.** Операнд может быть любым числом, кроме порядкового. Если операнд — массив чисел, операция выполняется над каждым из его элементов.

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

**Имя команды.** COMPLEMENT (COMP)

**Выполняемая операция.** Присвоение операнду знака, противоположного исходному.

**Формат.** ОЕ, ОА

**Операнды.** Операнд может быть любым числом, кроме порядкового. Если операнд — массив чисел, то операция выполняется над каждым его элементом.

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

**КОМАНДЫ СРАВНЕНИЯ И ПЕРЕХОДА****Имя команды.** EQUAL-BRANCH-FALSE (EQBF)

**Выполняемая операция.** При равенстве operandов по данной команде не выполняется никаких действий; в противном случае управление передается по указанному адресу команды.

**Формат.** 7, ОА, ОА, IA

**Операнды.** Операнды должны быть взаимно совместимыми (оии оба могут быть числами, строками символов, логическими величинами, указателями, полями и строками токеиов).

Если operandы — числа разного типа или размера, то перед их сравнением второй operand преобразуется в форму, совместимую с первым operandом. (Ошибка «переполнение» блокируется. В случае переполнения operandы считаются не равными друг другу.)

Если operandы — строки и (или) поля разной длины, то перед их сравнением более короткий operand дополняется пробелами (при работе с символьными данными) или нулями (при использовании токеиов). Эти дополнительные символы для строк или полей символов включаются справа, а для строк или полей токеиов — слева от исходных данных. Массивом может быть либо только первый operand, либо оба operandы. В последнем случае выполняется последовательное сравнение соответствующих элементов. Если первый operand — запись, второй должен быть записью с аналогичной структурой. При этом выполняется сравнение соответствующих компонентов. Операн-

ды-массивы (операнды-записи) считаются равными только в том случае, если все их соответствующие элементы (компоненты) равны. Для operandов-указателей осуществляется сравнение между собой только логических адресов (сравнение кодов доступа не производится).

*Ошибки.* Основной набор ошибок (за исключением ошибок «неверный тип операнда»), а также ошибки типа «недопустимая передача управления» и «трассировка».

#### **Имя команды. NOT-EQUAL-BRANCH-FALSE (NEBF)**

*Выполняемая операция.* Если operandы не равны, никаких действий не выполняется; в противном случае управление передается по указанному адресу команды.

*Формат.* 6, OA, OA, IA

*Operandы.* См. команду EQUAL-BRANCH-FALSE

*Ошибки.* См. команду EQUAL-BRANCH-FALSE

#### **Имя команды. LESS-THAN-BRANCH-FALSE (LTBF)**

*Выполняемая операция.* Если первый operand меньше второго, никаких действий не выполняется; в противном случае управление передается по указанному адресу команды.

*Формат.* 8, OA, OA, IA

*Operandы.* Operandы могут быть числами, строками или полями символов, строками или полями токенов. Если operandы — разные по типу или размеру числа, то перед их сравнением второй operand преобразуется в форму, совместимую с первым operandом. (Ошибка «переполнение» блокируется. В случае переполнения первый operand считается меньше второго.)

Сравнение строк или полей символов производится согласно правилам их представления в коде EBCDIC. Строки или поля токенов при сравнении рассматриваются как положительные шестнадцатеричные числа. Если длины сравниваемых строк или полей не совпадают, более короткие из них дополняются символами так же, как это делается при выполнении команды EQUAL-BRANCH-FALSE. Первый operand или оба operandы могут быть массивами. В последнем случае выполняется их поэлементное сравнение. При этом первый operand считается меньше второго только в том случае, когда все элементы первого operandна меньше соответствующих элементов второго operandia.

*Ошибки.* Основной набор ошибок, а также ошибки типа «недопустимая передача управления» и «трассировка».

#### **Имя команды. GREATER-THAN-BRANCH-FALSE (GTBF)**

*Выполняемая операция.* Если первый operand больше второго,

никаких действий не выполняется; в противном случае управление передается по указанному адресу команды.

**Формат.** 9, OA, OA, IA

**Операнды.** См. команду LESS-THAN-BRANCH-FALSE

**Ошибки.** См. команду LESS-THAN-BRANCH-FALSE

**Имя команды:** LESS-THAN-OR-EQUAL-BRANCH-FALSE (LEBF)

**Выполняемая операция.** Если первый operand меньше или равен второму, никаких действий не выполняется; в противном случае управление передается по указанному адресу команды.

**Формат.** A, OA, OA, IA

**Операнды.** См. команду LESS-THAN-BRANCH-FALSE

**Ошибки.** См. команду LESS-THAN-BRANCH-FALSE

**Имя команды:** GREATER-THAN-OR-EQUAL-BRANCH-FALSE (GEBF)

**Выполняемая операция.** Если первый operand больше или равен второму, никаких действий не выполняется; в противном случае управление передается по указанному адресу команды.

**Формат.** B, OA, OA, IA

**Операнды.** См. команду LESS-THAN-BRANCH-FALSE

**Ошибки.** См. команду LESS-THAN-BRANCH-FALSE

**Имя команды:** DEFINED-BRANCH-FALSE (DEFBF)

**Выполняемая операция.** Если значение operand'a определено, никаких действий не выполняется; в противном случае управление передается по указанному адресу команды.

**Формат.** 0004, OA, IA

**Операнды.** Operand может быть любого типа. Если operand — поле токенов, никаких действий не выполняется. В случае сложных данных (массива или записи) значение operand'a считается определенным только в том случае, когда заданы значения всех элементов. Если operand — поле символов, его значение считается определенным только в том случае, когда задано значение каждого элемента в этом поле. При использовании в качестве operand'a команды указателя его значение считается определенным, если в поле его представления отсутствует признак неопределенности и если объект, на который он ссылается, еще существует в системе.

**Ошибки.** Основной набор ошибок (за исключением ошибок «несовместимые operandы», «неопределенный operand» и «неверный тип operand'a»), а также ошибки типа «недопустимая передача управления» и «трассировка».

**Имя команды.** ITERATE

**Выполняемая операция.** Если первый операид меньше второго, значение первого операнда получает приращение и управление передается по указанному адресу команды; в противном случае никаких действий не выполняется.

**Формат.** 5, OA, OA, IA

**Операнды.** Оба операнда должны иметь одинаковый тип и могут быть целыми числами, в том числе увеличенной разрядности, а также порядковыми числами.

**Ошибки.** Основной набор ошибок, а также ошибки типа «недопустимая передача управления».

**Примечание.** Эту команду можно применять для организации цикла в качестве его завершающей команды.

**Имя команды.** ITERATE-REVERSE (ITERREV)

**Выполняемая операция.** Если первый операид больше второго, значение первого операнда уменьшается и управление передается по указанному адресу команды; в противном случае никаких действий не выполняется.

**Формат.** 03, OA, OA, IA

**Операнды.** Оба операнда должны иметь одинаковый тип и могут быть целыми числами, в том числе увеличенной разрядности, а также порядковыми числами.

**Ошибки.** Основной набор ошибок, а также ошибки типа «недопустимая передача управления».

**Имя команды.** CASE

**Выполняемая операция.** Передача управления по одному из указанных адресов команд в зависимости от значения операанда, являющегося целым или порядковым числом.

**Формат.** 0014, OA, X, IA0,...,IAx

**Операнды.** Операид может быть целым или порядковым числом. В поле непосредственных данных (X) длиной 2 токен указывается число от 1 до 255. Это число на единицу меньше числа возможных адресов команд, задаваемых в команде. Если значение операанда попадает в диапазон от 0 до X, управление передается команде с адресом IA<sub>i</sub>, где i — значение операанда. В противном случае управление передается команде с адресом IAx.

**Ошибки.** Основной набор ошибок (за исключением ошибок «несовместимые операнды»), а также ошибки типа «недопустимая передача управления» и «трассировка».

## ЛОГИЧЕСКИЕ КОМАНДЫ

### Имя команды. AND

**Выполняемая операция.** Логическое умножение (операция И) обоих операндов и запись результата на место первого операнда.

**Формат.** 05, ОА, ОА

**Операнды.** Операнды должны быть порядковыми числами со значениями 0 (ложно) и 1 (истинно). Первый операнд или оба операнда могут быть массивами.

**Ошибки.** Основной набор ошибок. Если значение порядкового числа не равно 0 или 1, регистрируется ошибка «неверный тип операнда».

### Имя команды. OR

**Выполняемая операция.** Логическое сложение (операция ИЛИ) обоих операндов и запись результата на место первого операнда.

**Формат.** 06, ОА, ОА

**Операнды.** См. команду AND

**Ошибки.** Основной набор ошибок.

### Имя команды. NOT

**Выполняемая операция.** Логическое отрицание (операция НЕ) операнда, т. е. изменение его значения на противоположное (значение «ложно» заменяется на «истинно», а значение «истинно» на «ложно»).

**Формат.** 0021, ОА

**Операнды.** Операнд может быть порядковым числом или массивом таких чисел.

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

## КОМАНДЫ ПОИСКА И МАНИПУЛИРОВАНИЯ СТРОКАМИ СИМВОЛОВ

### Имя команды. CONCATENATE (CONCAT)

**Выполняемая операция.** Подсоединение значения второго операнда к значению первого операнда.

**Формат.** 0D, ОА, ОА

**Операнды.** Первый операнд должен быть строкой и не может иметь неопределенного значения. Второй операнд должен быть строкой или полем того же типа. Длина первого операнда увеличивается на длину второго операнда, а значение второго операнда подсоединяется к концу первого операнда.

*Ошибки.* Основной набор ошибок, а также ошибки типа «переполнение».

### **Имя команды.** MOVE-SUBSTRING (MOVESS)

*Выполняемая операция.* Часть строки или поля (подстрока), указываемая второй группой операндов, пересыпается в подстроку, указанную первой группой операндов.

*Формат.* 04, ОА, ОА, ОА, ОА, ОА, ОА

*Операнды.* Первый и четвертый операнды должны быть строками или полями, причем совместными (оба операнда должны содержать либо символы, либо токены). Второй, третий, пятый и шестой операнды должны быть целыми числами. Операнды с первого по третий определяют подстроку, принимающую данные. Аналогично операнды с четвертого по шестой определяют подстроку, посыпающую данные. Второй операнд задает порядковый номер элемента, с которого начинается подстрока, а значение третьего операнда равно ее длине.

Если первый операнд является строкой, а не полем и  $L$  — ее текущая длина, то значение второго операнда должно быть большим или равным 1 и меньшим или равным  $L+1$ . В результате выполнения операции длина строки может увеличиваться. Если первый операнд является строкой с неопределенным значением, значение второго операнда должно равняться 1.

Если значение третьего операнда больше значения шестого операнда (принимающая подстрока длиннее посыпающей подстроки), происходит заполнение избыточной части подстроки одним из дополнительных символов. Для подстрок символов это дополнительное заполнение выполняется справа пробелами, а для подстрок токенов — слева нулями. Указанное заполнение имеет место только в пределах подстроки-результата, а не во всей принимающей строке или поле.

Если обе подстроки принадлежат одной и той же строке или полю и взаимно перекрываются, результат соответствует таким действиям, при которых посыпаемая подстрока сначала извлекается из того места, где находится, а затем помещается в принимающую подстроку.

*Ошибки.* Основной набор ошибок, а также ошибки типа «переполнение».

### **Имя команды.** INDEX

*Выполняемая операция.* Начиная с указанной позиции по строке или полю, выполняется поиск заданной подстроки. Если подстрока будет найдена, первый операнд будет содержать индекс (порядковый номер) первого элемента этой подстроки в строке или поле. Если подстрока обнаружена не будет, значение первого операнда полагается равным 0.

**Формат. 07, ОА, ОА, ОА, ОА**

**Операнды.** Первый операнд должен быть целым числом. Перед началом выполнения команды его значением является порядковый номер элемента в строке, с которого должен быть начат поиск. Второй операнд также должен быть целым числом. Если он имеет ненулевое значение, оно определяет порядковый номер элемента в строке, где поиск должен быть прекращен. Если его значение равно 0, поиск будет проводиться до конца строки. Третий операнд задает строку или поле, где производится поиск. Четвертый операнд должен быть строкой или полем и иметь тот же тип (т. е. быть строкой или полем символов, логических величин или токенов), что и третий операнд. Четвертый операнд — это подстрока, местоположение копии которой в третьем операнде подлежит обнаружению.

**Ошибки.** Основной набор ошибок. В нижеследующих случаях регистрируется ошибка «выход индекса за допустимые границы»: 1) порядковый номер элемента, с которого должен быть начат поиск, меньше 1; 2) порядковый номер элемента, где поиск должен быть прекращен, отрицательный; 3) сумма значений порядкового номера элемента, где поиск должен быть начат или закончен, и длины подстроки, местоположение которой определяется, превышает увеличенную на 1 единицу (размер) строки (поля), в которой ведется поиск.

**Имя команды. LENGTH**

**Выполняемая операция.** Значение длины второго операнда (строки или поля) присваивается первому операнду.

**Формат. 08, ОА, ОА**

**Операнды.** Первый операнд должен быть целым числом, а второй — строкой или полем.

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые операнды» и «неопределенный операнд».

**Имя команды. SEARCH**

**Выполняемая операция.** В команде задаются массив (или срез массива), значение индекса, точка окончания и значение ключа для поиска. Выполняется поиск заданного значения ключа среди всех элементов массива, начиная с элемента с указанным индексом. При обнаружении такого элемента его индекс присваивается операнду, в котором первоначально указывается исходное значение индекса. Если элемент обнаружить не удается, упомянутому операнду присваивается нулевое значение.

**Формат. 0003, ОА, ОА, ОА, ОА**

**Операнды.** Первый операнд должен быть целым числом. Первоначально в нем содержится значение индекса элемента, с которого следует начинать поиск. По окончании операции в нем

располагается порядковый номер элемента, значение которого совпадает с заданным ключевым (или равно 0). Второй операнд должен быть целым числом. Если оно не равно 0, то является индексом того элемента в массиве, на котором поиск должен быть закончен. Если его значение равно 0, поиск ведется до последнего элемента массива. Третий операнд должен быть одномерным массивом или срезом (адрес операнда задается в форме адреса массива или среза). Четвертый операнд содержит ключ — значение, поиск которого проводится. Элементы массива сравниваются с ключевым значением по правилам, описанным для команды EQBF.

**Ошибки.** Основной набор ошибок.

## КОМАНДЫ УПРАВЛЕНИЯ

### Имя команды. CALL

**Выполняемая операция.** Прерывание выполнения данного модуля и инициирование выполнения другого модуля с заданной точки входа. Для вызываемого модуля выполняются выделение и начальная установка памяти для динамической части адресного пространства.

**Формат.** D, OA, X, A1, OA1,...,Ax, OAх

**Операнды.** Первый operand — указатель на точку входа модуля (должен содержать разрешение на чтение). Значение указателя должно предварительно формироваться с помощью команды CREATE-ENTRY-CAPABILITY. Поле непосредственных данных имеет длину 2 токен и содержит шестнадцатеричное число X, равное количеству передаваемых фактических параметров. X пар полей, следующих за этими двумя токенами, характеризуют отдельные параметры. Ai является непосредственным адресом (длиной 1 токен), значение которого определяет, передается ли фактический параметр с разрешением как чтения, так и записи (значение=0011) или только чтения (значение=0111). OAi обозначает адрес фактического параметра в форме адреса операнда. Фактическими параметрами не могут быть литералы или срезы, а также адреса ячеек «косвенный доступ к данным».

**Ошибки.** Основной набор ошибок, а также ошибки типа «трасировка команды CALL», «недопустимая передача управления» и «недостаточный объем памяти». Если в программе обработки ошибок встречается команда CALL и существуют области активации, подчиненные текущей области (образованные после данной области), все они ликвидируются.

**Примечания.** При выполнении команды CALL действительной передачи фактических параметров, соответствующих формальным параметрам вызываемого модуля, не происходит. Эта

операция выполняется по команде ACTIVATE при вызываемой точке входа. По команде CALL создается запись активации, помещаемая как последняя запись в стек записей активации данного процесса.

#### **Имя команды.** ACTIVATE (ACT)

**Выполняемая операция.** Проверка формальных и фактических параметров на соответствие друг другу и присвоение начальных значений указанным формальным параметрам.

**Формат.** С, X, CA1,...,CAx

**Операнды.** В поле непосредственных данных (длиной 2 токен) в шестнадцатеричной форме задается количество параметров (X). Следующие за этим полем X полей являются адресами ячеек «параметр». Этим формальным параметрам присваиваются значения фактических параметров, переданные последней выполнявшейся в данной процесс-машине командой CALL или LCALL.

**Ошибки.** Имеют место ошибки «неправильная адресация», «даные неизвестного формата», «имитация ошибки», «неверный тип операнда», «неправильное число передаваемых данных», «несовместимые операнды».

**Примечания.** Правила совместимости формальных и фактических параметров приведены в табл. 15.2. Команда ACTIVATE не обязательно должна быть первой командой у заданной точки входа в модуль, но она должна предшествовать любому обращению к ячейкам «параметр». (В противном случае параметр будет иметь неопределенное значение.) Если команда ACTIVATE подлежит включению в группу команд, связанных с вызовом модуля, она должна предшествовать любой команде, определяющей фактические параметры (CALL, LCALL, SEND, RECEIVE).

#### **Имя команды.** RETURN

**Выполняемая операция.** Завершение выполнения данного модуля и передача управления команде, следующей за командой CALL, которая вызывает данный модуль.

**Формат.** ОА

**Операнды.** Отсутствуют.

**Ошибки.** Имеют место ошибки типа «имитация ошибки».

**Примечания.** Команда RETURN отменяет режим, установленный предыдущими командами CALL и ACTIVATE, т. е. уничтожает текущую запись активации. Если это была единственная запись активации, то уничтожается и данная процесс-машина.

**Таблица 15.2.** Правила соответствия между формальными и фактическими параметрами

Формальный параметр	Фактический параметр
Параметр с динамически определяемым типом Целое число, указатель, целое число увеличенной разрядности, порядковое число	Любая ячейка простого типа, кроме ячейки, тип которой определяется пользователем По типу идентичный формальному параметру
Десятичное число с фиксированной или плавающей точкой, логическая величина, поле (строка) символов или токенов	По типу идентичный формальному параметру; должна быть идентичность и по размеру, за исключением тех случаев, когда формальный параметр имеет динамически определяемый размер
Запись	Запись с таким же числом компонентов. Тип и размер каждого компонента должны совпадать с типом и размером соответствующих компонентов записи — формального параметра. Если компонентами записи являются массивы, у них должна быть такая же размерность, как и у соответствующих массивов-компонентов в записи — формальном параметре, и идентичные атрибуты элементов. Они должны также иметь одинаковые верхние границы для индексов, если только массивы в записи — формальным параметре не являются массивами с динамическим определением границ
Массив	Массив такой же размерности. Если массив — формальный параметр не является массивом с динамическим определением границ, последние должны быть идентичными для обоих массивов. Типы элементов в обоих массивах также должны быть назначены одинаковыми, если только массив — формальный параметр не является массивом с динамическим определением типа. То же относится и к их атрибуту «размер»
Параметр, тип которого определяется пользователем	Аналогичные атрибуты пользователя, а также совместимость по вышеприведенным правилам для ячеек, определяемых формальными и фактическими параметрами

### Имя команды. LOCAL-CALL (LCALL)

**Выполняемая операция.** Приостановка последовательного выполнения команд и передача управления указываемой команде в данном модуле.

**Формат.** 000B, IA, X, A1, OA1,...,Ax, OAx

**Операнды.** В первом поле адреса задается адрес команды, которой передается управление. Остальные поля — такие же, как

и в команде CALL. Передаваемые фактические параметры не могут быть литералами или косвенными указателями.

**Ошибки.** Основной набор ошибок, а также ошибки типа «недопустимая передача управления», «страсировка команды CALL» и «недостаточный объем памяти».

**Примечания.** В отличие от команды CALL команда LCALL не создает записи активации. Следовательно, внутренние процедуры не допускают рекурсивного обращения (если только компилятор не генерирует команду ALLOCATE в целях имитации наличия записи активации); согласование всех имен возлагается на компилятор.

**Имя команды.** LOCAL-RETURN (LRETURN)

**Выполняемая операция.** Передача управления команде, следующей за последней выполнившейся в данном модуле командой LCALL.

**Формат.** 000C

**Операнды.** Отсутствуют.

**Ошибки.** Имеют место ошибки типа «недопустимая передача управления» (если не было предшествующей команды LCALL) и «имитация ошибки».

**Примечания.** Если команда LRETURN встречается в программе обработки ошибок, а в модуле отсутствует команда LCALL, выполнение программы обработки ошибок прекращается и управление передается команде, при выполнении которой произошла регистрация ошибки<sup>1)</sup>.

**Имя команды.** BRANCH (B)

**Выполняемая операция.** Передача управления по указанному адресу команды.

**Формат.** E, IA

**Ошибки.** Имеют место ошибки типа «недопустимая передача управления» и «имитация ошибки».

## КОМАНДЫ АДРЕСАЦИИ

**Имя команды.** COMPUTE-CAPABILITY (CCAP)

**Выполняемая операция.** Запись потенциального адреса второго операнда на место первого операнда.

**Формат.** 0010, OA, OA

**Операнды.** Первый operand должен быть указателем. Второй operand может быть любым, кроме ячейки «параметр» лексемы или всего среза. В коде доступа потенциального адреса

<sup>1)</sup> Или следующей за ней команде, если она была командой RAISE-FAULT. — Прим. перев.

устанавливаются разрешение на копирование и запрет на уничтожение. Разрешение на чтение-запись задается в соответствии с текущим кодом доступа данного модуля ко второму операнду.

Потенциальный адрес является адресом ячейки<sup>1)</sup>, указываемой вторым операндом. Если такая ячейка — «косвенный доступ к данным», то на место первого операнда записывается указатель, соответствующий этой ячейке. При этом код доступа указателя должен содержать разрешение на копирование.

*Ошибки.* Основной набор ошибок, за исключением ошибок типа «неопределенный операнд» и «несовместимые операнды».

#### **Имя команды. COMPUTE-INDIRECT-CAPABILITY (CICAP)**

*Выполняемая операция.* Запись косвенного потенциального адреса второго операида на место первого операида.

*Формат.* 0016, ОА, ОА

*Операнды.* Оба операида должны быть указателями. Первый операид становится косвенным потенциальным адресом второго операида. Его код доступа дублирует код доступа второго операида. В коде доступа второго операида должно быть разрешение на копирование, а сам операид не может быть косвенным потенциальным адресом или параметром.

*Ошибки.* Основной набор ошибок, за исключением ошибок типа «несовместимые операнды» и «неопределенный операид».

#### **Имя команды. COMPUTE-CAPABILITY-EXTERNALLY (CCAPEX)**

*Выполняемая операция.* Запись на место первого операида потенциального адреса операида, к которому имеет место внешняя адресация с помощью второго, третьего и четвертого операида (адресация к операиду называется внешней, поскольку он принадлежит другому адресному пространству заданного модуля данной процесс-машины).

*Формат.* 0012, ОА, ОА, ОА, ОА

*Операнды.* Первые три операида должны быть указателями. Второй операид представляет собой потенциальный адрес процесс-машины. Третий операид является потенциальным адресом объекта «модуль». Четвертый операид должен быть строкой или полем токениев. Он выполняет такую же роль, как адрес второго операида в команде COMPUTE-CAPABILITY. Этот операид [в форме адреса ячейки, массива (реза), элемента мас-

<sup>1)</sup> Логическим или системным адресом ячейки в отличие от адреса ячейки как разновидности адреса операндов, используемого в командах.—Прим. перев.

сива (реза), запись или компонента записи] используется как адрес операнда, входящего в модуль, на который ссылается третий operand.

Коды доступа, которые следует задать в исходных потенциальных адресах, и код доступа, который будет получен с результатирующим потенциальным адресом, зависят как от того, принадлежит ли operand динамической или статической части адресного пространства адресуемого модуля, так и от того, является ли он косвенным доступом к данным. Возможны следующие варианты:

1. Operand относится к статической части адресного пространства и не является косвенным доступом к данным. В этом случае система не пользуется потенциальным адресом процессор-машины и никаких специальных требований к коду доступа потенциального адреса не предъявляется. Код доступа потенциального адреса-результата содержит разрешение на копирование и запрет на уничтожение. Разрешение на чтение-запись в коде доступа результата устанавливается в соответствии с директивной информацией в коде доступа потенциального адреса модуля.

2. Operand относится к динамической части адресного пространства и не является косвенным доступом к данным. Отсутствуют какие-либо специальные требования к кодам доступа потенциальных адресов как процессор-машины, так и модуля. Код доступа потенциального адреса-результата содержит разрешение на копирование и запрет на уничтожение. Разрешение на чтение-запись в коде доступа результата устанавливается в соответствии с директивной информацией в коде доступа потенциального адреса модуля.

3. Operand является косвенным доступом к данным и относится к статической части адресного пространства. В этом случае система также не пользуется потенциальным адресом процессор-машины. Потенциальный адрес модуля должен содержать разрешение только на чтение, а указатель, соответствующий ячейке «косвенный доступ к данным», — разрешение на копирование. Содержимое этого указателя определяет значение потенциального адреса результата.

4. Operand является косвенным доступом к данным и относится к динамической части адресного пространства. Потенциальный адрес процессор-машины должен содержать разрешение только на чтение, а потенциальный адрес указателя — разрешение на копирование. Код доступа в потенциальном адресе модуля не нуждается в задании какой-либо специальной директивной информации. В ячейку результата помещается содержимое указателя, соответствующего ячейке «косвенный доступ к данным».

Если операнд или указатель, соответствующий ячейке «косвенный доступ к данным», которая выполняет роль операнда, принадлежит динамической части адресного пространства, результат определяется текущим (последним) обращением к этому модулю при работе указанной процесс-машины.

**Ошибки.** Основной набор ошибок.

**Примечания.** В семантическом отношении выполнение команды ССАРЕХ полностью эквивалентно выполнению команды ССАР в заданном модуле указанной процесс-машины. Команда ССАРЕХ разрабатывалась для создания средств отладки, с помощью которых можно было бы производить обращение к данным отлаживаемой программы.

Хотя в последнем операнде команды адрес операнда представляет собой обращение к адресному пространству другого модуля, длина этого адреса должна соответствовать величине CAS в модуле, содержащем данную команду ССАРЕХ. Команда ССАРЕХ может использоваться совместно с командой DOT для определения текущих атрибутов данных из другого модуля. В таком случае указатель, в который будет загружаться результат, должен соответствовать ячейке «косвенный доступ к данным» с динамически определяемым типом; последняя должна являться операндом команды DOT. Если атрибуты ячейки, потенциальный адрес которой определяется, неизвестны, в четвертом операнде команды ССАРЕХ должна быть использована форма адресации, которую следовало бы применить для массива, имеющего размерность 15.

### **Имя команды. CHANGE-ACCESS (CACC)**

**Выполняемая операция.** Установление дополнительных ограниченных в коде доступа указываемого операнда (изменение значения кода на новое значение, задаваемое в команде).

**Формат.** 0006, X, ОА

**Операнды.** Операнд должен быть указателем. Поле непосредственных данных (X) имеет длину 1 токен. Новое значение кода доступа формируется в результате выполнения логической операции ИЛИ над старым значением кода доступа и величиной X. Так вводится специальная директивная информация (дополнительные ограничения) в код доступа указателя.

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

### **Имя команды. ALLOCATE (ALLOC)**

**Выполняемая операция.** Создание объекта «память данных» с описанием возможного содержимого, задаваемого операндом команды.

**Формат. 0020, X, ОА**

**Операнды.** В последнем разряде поля непосредственных данных (X) длиной 1 токен указывается, должен ли данный объект автоматически ликвидироваться при уничтожении процесс-машины, выполняющей в данный момент эту команду. Значение xxxx является требованием на уничтожение.

В третьем разряде этого поля указывается, должно ли присваиваться выделенной памяти в качестве начального неопределенное значение. Требованию такой начальной установки соответствует содержимое поля, равное xx0x; содержимое xx1x означает отсутствие этого требования. Если требование начальной установки отсутствует, а создаваемый объект должен содержать ячейки «указатели», отсутствие этого требования игнорируется и всем указателям присваивается неопределенное значение.

Адрес операнда должен быть адресом ячейки, элемента массива, массива или записи. Операнд должен быть ячейкой «косвенный доступ к данным», которая может содержать описание ячейки любого типа. Косвенный доступ к данным не может располагать атрибутами, обеспечивающими динамическое определение размера или типа.

Если операндом является массив, ссылка к которому делается в форме адреса массива, содержимое полей верхней границы значения индекса по соответствующей размерности массива используется для определения размера массива. Если же адресация к массиву задается посредством адреса элемента массива, содержимое полей верхней границы значения индекса по соответствующей размерности массива в его теге должно равняться 0 (т. е. массив должен иметь динамически определяемые границы). В этом случае текущие значения индексов считаются верхними значениями индексов при выделении памяти под массив.

Описание возможных ячеек создаваемого объекта содержится во вложенном теге (включающем соответствующие теги). Код доступа в указателе, задаваемом соответствующей ячейкой «косвенный доступ к данным», имеет значение 0000 (т. е. санкционированы чтение, запись, уничтожение и копирование), а логический адрес в нем ссылается на генерируемый объект «память данных».

**Ошибки.** Основной набор ошибок (за исключением ошибок «несовместимые операнды»), а также ошибки типа «недостаточный объем памяти».

**Примечание.** Учитывая особенности архитектуры, при выполнении команды ALLOCATE рекомендуется задавать ее необязательный параметр «присвоение неопределенного значения в качестве начального». Возможность использования этого парамет-

ра как необязательного была введена для удобства разработки программ компиляторов, в которых после команды ALLOCATE предполагается выполнение начальной установки программным путем.

### *Имя команды. DESTROY*

*Выполняемая операция. Уничтожение объекта, адресуемого операндом.*

*Формат. 0007, ОА*

*Операнды.* Операнд должен быть указателем, косвенным доступом к данным или параметром-указателем. По данной команде выполняется ликвидация объекта, адресуемого указателем-операндом (или указателем, соответствующим ячейке «косвенный доступ к данным»). Кодом доступа указателя должна быть санкционирована возможность уничтожения объекта. После выполнения операции в указатель записывается признак неопределенного значения. Если команда DESTROY относится к модулю, находящемуся в активном состоянии (для которого существует область активации), немедленно выполняется уничтожение системного имени этого модуля как объекта (что означает невозможность дальнейших обращений к нему, например посредством команды CALL). Однако сам объект будет выведен из системы лишь после того, как прекратят существование все его записи активации. Если уничтожению подлежит объект «порт» с ожидающими обработки запросами команд SEND или RECEIVE, порт уничтожается, а выполнение указанных «повисших» команд прерывается с регистрацией ошибки «недействительный указатель»<sup>1)</sup>.

Если объектом команды DESTROY является процесс-машина, содержащая объекты с признаком их ликвидации косвенным путем при уничтожении этой процесс-машины, эти объекты уничтожаются вместе с процесс-машиной. (См. описание передачи АМ в гл. 9.)

Требование уничтожения объекта «память данных» также может выполняться с задержкой, предусмотренной для того, чтобы система не уничтожила объект в момент обращения к нему команды из другой процесс-машины.

*Ошибки.* Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

### *Имя команды. CHANGE-LOGICAL-ADDRESS (CLA)*

*Выполняемая операция. Присвоение указанному объекту ново-*

<sup>1)</sup> По-видимому, разновидность состояния «недействительный потенциальный адрес». — Прим. перев.

го логического адреса и уничтожение его старого логического адреса.

**Формат.** 0022, ОА

**Операнды.** Операндом должна быть ячейка «указатель» или «косвенный доступ к данным». Код доступа в указателе должен содержать директивную информацию, санкционирующую чтение, запись, копирование и уничтожение. При этом указатель должен ссылаться на весь объект, а не на какую-либо его часть. В результате выполнения команды объекту, на который ссылается указатель, назначается новый логический адрес, помещаемый в тот же указатель с кодом доступа, санкционирующим выполнение перечисленных операций. Любая последующая попытка воспользоваться старым логическим адресом объекта приводит к регистрации ошибки «недействительный указатель».

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

**Имя команды.** CREATE-MODULE (CMODULE)

**Выполняемая операция.** Создание объекта «модуль».

**Формат.** 0009, X, ОА, ОА

**Операнды.** Содержимое поля непосредственных данных (X) длиной 1 токен показывает, должен ли объект «модуль» автоматически ликвидироваться при уничтожении процесс-машины, в которой выполняется эта команда. В частности, если содержимое этого поля равно xxx0, то требуется ликвидация модуля при уничтожении процесса. Порядок уничтожения модуля приводится в описании команды DESTROY.

Первый operand должен быть указателем, а второй — строкой или полем токенов. Содержимое этого поля или строки должно соответствовать формату внешнего модуля (см. рис. 14.3). Машина проверяет правильность формата модуля, копирует его во внутреннюю память и загружает в указатель потенциальный адрес объекта «модуль» с полным набором санкционированных возможностей доступа (чтение, запись, уничтожение и копирование). Для всех параметров и указателей в модуле устанавливается признак неопределенного значения. В рамках сформированного объекта «модуль» выделяется память для массивов, описываемых в статической части адресного пространства.

**Ошибки.** Основной набор ошибок (за исключением ошибок типа «несовместимые операнды»), а также ошибки «недействительный модуль» и «недостаточный объем памяти». При регистрации ошибки «недействительный модуль» пятый фактический параметр, передаваемый программе обработки ошибок при его-

вызове, содержит информацию о характере ошибки. Этот параметр может иметь следующие значения:

000001 — ошибка в указателях в заголовке модуля;

000002 — недопустимые значения в полях CAS, IAS или SIS;

cccc03 — недопустимая к использованию ячейка или неправильная связь ячеек в модуле;

iiii04 — адрес операида в некоторой команде не соответствует началу ячейки или ее компонента.

Здесь cccc — четыре младших токена адреса ячейки, в которой обнаружена ошибка; iiii — четыре младших токена адреса неправильной команды.

#### **Имя команды. COMPUTE-ENTRY-CAPABILITY (CECAP)**

**Выполняемая операция.** Вычисление потенциального адреса для указываемой команды в заданном модуле.

**Формат.** 000F, OA, OA, OA

**Операнды.** Первый operand является указателем, используемым для размещения результата. Второй operand — указатель требуемого объекта «модуль». Третий operand представляет собой поле длиной 5 токен, где задается адрес команды модуля, для которой необходимо вычислить потенциальный адрес. Этот адрес помещается в первый operand. Код доступа санкционирует чтение и копирование.

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые operandы». Ошибка «неправильная адресация» регистрируется в тех случаях, когда адрес команды выходит за пределы области команд. Ошибка «нарушение защиты доступа» регистрируется тогда, когда кодом доступа во втором operandе не предусмотрены возможности чтения или копирования.

#### **Имя команды. LINK**

**Выполняемая операция.** Присвоение потенциального адреса.

**Формат.** 000A, OA, OA, OA

**Операнды.** Первый operand является указателем объекта «модуль». В указателе должна быть предусмотрена возможность записи. Второй operand есть поле из 5 токен. В этом поле задается адрес ячейки, принадлежащий статической части адресного пространства модуля. Третий operand является указателем. Значение этого указателя присваивается указателю, определяемому первым и вторым operandами.

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые operandы». Если ячейка для размещения результата находится в статической части адресного пространства или не является указателем, то регистрируется ошибка «неправильная адресация».

**Имя команды. DESCRIBE-CAPABILITY**

**Выполняемая операция.** Определение по заданному указателю его значения и данных об адресуемой им ячейке.

**Формат.** 001B, OA, OA, OA, OA, OA

**Операнды.** Первые два операнда должны иметь целочисленные значения. Третий operand — указатель. Четвертый operand — одномерный массив полей символов размера 6. (Адрес операнда должен быть представлен в форме адреса массива.) Массив должен состоять по крайней мере из трех элементов. Пятый operand является указателем, по которому должна быть получена информация.

По данной команде в первых трех операндах записывается информация о значении указателя и адресуемой им ячейке. Информация, получаемая при выполнении команды, представлена в табл. 15.3, где OA4(i) — значение i-го элемента четвертого операнда.

**Ошибки.** Основной набор ошибок.

**Таблица 15.3. Результаты выполнения команды DESCRIBE-CAPABILITY**

Объект, адресуемый указателем	Тип информации в ячейках, адресуемых					
	первым operandом OA1	вторым operandом OA2	третьим operandом OA3	четвертым operandом		
	OA4 (1)	OA4 (2)	OA4 (3)			
Модуль	1	15	16	6	7	8
Порт	1	2	16	6	7	9
Память данных	1	5	16	6	7	10
Процесс-машина	13	14	16	6	7	11
Ячейка в статической части адресного пространства модуля	3	5	17	6	7	12
Ячейка в записи активации	3	5	17	6	7	12
Ячейка в памяти данных	3	5	17	6	7	12
Точка входа в модуль	4	5	17	6	7	12
Внешний порт	5	5	5	6	7	12

**Примечание.** В таблице использованы следующие обозначения:

- 1 — размер объекта (в единицах АМ);
- 2 — количество процесс-машины, находящихся в данный момент в очереди к порту;
- 3 — тип ячейки (значение первых четырех двоичных разрядов тега);
- 4 — адрес точки входа в форме адреса команды;
- 5 — неопределенное значение;
- 6 — директивная информация о видах доступа, предоставленного данному указателю;

Номер символа	Значение символа
1	Пробел или R (чтение)
2	Пробел или W (запись)
3	Пробел или D (уничтожение)
4	Пробел или C (копирование)
5	Тип указателя: пробел (прямой указатель) или I (косвенный указатель)
6	Пробел

7 — тип объекта, адресуемого указателем. Тип кодируется первыми двумя символами согласно следующим обозначениям:  
 МО — объект «модуль»;  
 РО — объект «порт»;  
 ДО — объект «память данных»;  
 ХО — процесс-машина;  
 МС — ячейка в статической части адресного пространства модуля;  
 АС — ячейка в записи активации;  
 DC — ячейка в памяти данных;  
 МЕ — точка входа в модуль;  
 ЕО — внешний порт.

Символы с 3-го по 6-й являются пробелами:

8 — состояние модуля:

Номер символа	Значение символа
1	Пробел или Р (подлежит ликвидации при уничтожении процесс-машины)
2	Пробел или А (в активном состоянии)
3	Пробел или G (в состоянии «сохраняем»)
4	Пробел или Т (выполняется трассировка)
5—6	Пробелы

9 — состояние порта:

Номер символа	Значение символа
1	Пробел или Р (подлежит ликвидации при уничтожении процесс-машины)
2	Пробел, или S (невыполненная передача), или R (не выполненный прием)
3—6	Пробелы

10 — состояние памяти данных:

Номер символа	Значение символа
1	Пробел или Р (подлежит ликвидации при уничтожении процесс-машины)
2—6	Пробелы

## 11 — состояние процесс-машины:

Номер символа	Значение символа
1	Пробел или P (подлежит ликвидации при уничтожении процесс-машины)
2	Состояние машины: A (активна) B (заблокирована по команде GUARD) P (ожидает обмена с портом) D (приостановлена по команде DELAY) S (остановлена/ожидает перехода в это состояние) X (ожидает обмена с внешним портом)
3—6	Пробелы

12 — пробел;  
 13 — размер свободной памяти (в единицах АМ), которой в данный момент располагает процесс-машина;  
 14 — приоритет процесс-машины;  
 15 — количество процесс-машины, блокируемых модулем, находящимся в состоянии «сохраняется»;  
 16 — потенциальный адрес процесс-машины, создавшей данный объект, при условии, что последний подлежит автоматическому уничтожению с уничтожением процесс-машины; иначе — неопределенное значение. В коде доступа потенциального адреса имеется разрешение только на копирование;  
 17 — потенциальный адрес того объекта (модуля и памяти данных), которому принадлежит адресуемый элемент. Если адресуется ячейка из записи активации, выдается потенциальный адрес соответствующего модуля; в коде доступа этого адреса имеется разрешение только на копирование.

## КОМАНДЫ УПРАВЛЕНИЯ ПРОЦЕСС-МАШИНАМИ

**Имя команды.** CREATE-PROCESS-MACHINE (CMACHINE)  
**Выполняемая операция.** Создание процесс-машины и передача ей имеющейся в наличии памяти (измеряемой количеством АМ) от текущей процесс-машины. Работа новой процесс-машины начинается с определенной точки входа указываемого модуля, для данных которого, описанных в динамической части его адресного пространства, осуществляется выделение памяти с последующим присвоением ей начальных значений.

**Формат.** 001A, X, OA, OA, OA, OA

**Операнды.** Содержимое поля непосредственных данных (X) длиной 1 токен указывает, должна ли новая процесс-машина автоматически уничтожаться при ликвидации процесс-машины, в которой выполняется данная команда. В частности, значение xxx0 указывает на необходимость уничтожения.

Первый operand должен быть указателем. В него помещается потенциальный адрес создаваемой машины с директивной информацией, санкционирующей в коде доступа чтение, запись, копирование и уничтожение. Второй operand должен быть ука-

зателем (с санкционированным чтением) на точку входа в модуль. Третий операнд должен иметь целочисленное значение, задающее объем имеющейся в распоряжении памяти (измеряемой количеством АМ), который должен быть передан созданной процесс-машине от данной процесс-машины (см. следующий раздел).

Четвертый operand определяет единственный фактический параметр, который может быть передан при вызове модуля по указанной точке входа. Этот operand должен быть указателем с санкционированием копирования в коде доступа. Созданная новая машина не обеспечивается адресацией к данному указателю. Вместо этого в новой машине создается копия его значения, и процесс, выполняемый в рамках новой машины, может обращаться к этому значению указателя посредством ячейки «параметр».

Текущая процесс-машинна продолжает свою работу с команды, следующей за командой CMACHINE. Приоритет новой машины полагается равным приоритету текущей машины.

**Ошибки.** Основной набор ошибок (за исключением ошибок «несовместимые operandы»), а также ошибки типа «трассировка» и «недостаточный объем памяти». Последний тип ошибок регистрируется в тех случаях, когда текущая машина не имеет в распоряжении достаточного количества АМ (свободной памяти) для создания новой процесс-машины или когда для передачи новой машине запрошено большее число АМ, чем имеется в наличии.

### Имя команды. TRANSFER-AM (TRANSAM)

**Выполняемая операция.** Пересылка части свободной памяти между двумя заданными процесс-машинами.

**Формат.** 001E, OA, OA, OA

**Операнды.** Первые два операнда должны быть указателями, адресующимися к двум процесс-машинам (один из них может адресоваться к текущей процесс-машине). Третий operand должен быть целым (положительным) числом, определяющим количество АМ (единиц свободной памяти), которое подлежит пересылке от процесс-машины, указанной вторым operandом, процесс-машине, задаваемой первым operandом.

В коде доступа второго указателя должна быть санкционирована запись; какие-либо требования к коду доступа первого указателя не предъявляются.

**Ошибки.** Основной набор ошибок (за исключением ошибок «несовместимые operandы»), а также ошибки типа «недостаточный объем памяти». Ошибки последнего типа регистрируются в тех случаях, когда количество АМ (свободной памяти) в про-

цессе, от которого она должна браться, меньше запрошенной величины.

**Примечание.** Объем памяти, занимаемый объектом, измеряется единицей, называемой АМ. Свободная память, измеряемая так называемыми *наличными* (имеющимися в распоряжении) АМ, является атрибутом процесс-машины и определяет потенциальные возможности выделения памяти данным процессом.

**Имя команды.** CONTROL-PROCESS-MACHINE (ХМАШИНЕ)

**Выполняемая операция.** Эта команда обладает следующими возможностями: 1) перевода указанной процесс-машины в состояние останова, 2) перевода процесс-машины из состояния останова в активное состояние, 3) имитации ошибки для указанной машины и 4) смены значения приоритета указанной машины.

**Формат:** 001F, X, ОА, ОА

**Операнды.** Содержимое поля непосредственных данных (X) длиной 1 токен определяет, какую именно операцию из числа возможных выполняет команда:

Значение Х	Выполняемая операция
1	Перевод в состояние останова
2	Перевод в активное состояние
3	Имитация ошибки
4	Смена значения приоритета

Первый operand должен быть указателем, содержащим потенциальный адрес (с санкционированием запин) требуемой процесс-машины; второй operand — целое число.

При X=1 указанная процесс-машина переводится в состояние останова по окончании выполнения очередной команды. В состоянии останова никакие команды процесс-машины не реализуются; выполнение команд может быть продолжено после перевода процесс-машины в активное состояние. Процесс-машины, находящиеся в состоянии блокировки (по команде GUARD) или ожидания (выполнения обмена с внутренним или внешним портом машины), не могут быть сразу же переведены в состояние останова. В этих случаях при попытке перевести их в указанное состояние они переходят в состояние «ожидание перехода в останов». Фактически процесс-машины будут переведены в останов сразу же при выходе из состояния блокировки или ожидания. При X=1 значение второго operand не существенно.

При  $X=2$  указанная процесс-машина переводится из состояния останова или состояния «ожидание перехода в останов» в активное состояние. Если машина находилась в активном состоянии, состоянии блокировки или состоянии, при котором выполняемый ею модуль «охраняется», то по данной команде никакие действия не выполняются. При  $X=2$  значение второго операнда также не существенно.

При  $X=3$  для указанной процесс-машины моделируется состояние ошибки. При этом процесс-машина выводится из любого состояния, в каком бы она ни находилась, и переводится в активное состояние. Значение второго операнда передается в качестве пятого фактического параметра кода ошибок (с преобразованием в поле длиной 6 токен).

При  $X=4$  выполняется изменение приоритета для указанной процесс-машины. В качестве нового значения приоритета берется наименьшее из двух возможных: значение второго операнда или значение приоритета машины, выполняющей данную команду.

**Ошибки.** Основной набор ошибок (за исключением ошибок «несовместимые операнды»), а также ошибки типа «неверный код операции» (в случае если  $X$  не равно 1, 2, 3 или 4).

**Примечание.** Понятие приоритетов, используемое при конкретной реализации вычислительной системы, не предопределется архитектурой системы SWARD. При работе с приоритетами следует учитывать, что большие числовые значения соответствуют более высоким приоритетам.

#### **Имя команды.** DESCRIBE-PROCESS-STACK (DSTACK)

**Выполняемая операция.** Описание структуры стека активных модулей для заданной процесс-машины.

**Формат.** 0011, OA, OA, OA, OA

**Операнды.** Первый operand должен быть одномерным массивом, каждый элемент которого является двухкомпонентной записью: один компонент — поле из 5 токен, другой — указатель (OA1::=ca\*000).

Второй operand должен быть целым числом, третий — указателем, четвертый operand — указателем, адресующим к необходимой процесс-машине. Специальных требований к коду доступа этого указателя не предъявляется. После выполнения команды массив содержит информацию о состоянии процесса указанной процесс-машины. Информация о каждой активации какого-либо модуля этой машины регистрируется в виде отдельного элемента массива: первый элемент — текущая активация, второй элемент — предшествующая и т. д. В каждом элементе массива поле токенов содержит адрес следующей команды, подлежащей выполнению, а указатель ссылается на модуль,

соответствующий этой активации (в коде доступа указана возможность только копирования). Значение второго операнда определяет количество активных модулей в процесссе.

Если число элементов в массиве меньше числа активных модулей, массив содержит информацию только о самых последних (верхних в стеке) модулях активации. Если число элементов в массиве превышает число активных модулей, содержимое оставшихся элементов массива сохраняется без изменения.

Если процесс-машина находится в состоянии блокировки, ожидания или ожидания перехода в останов (см. описание команды DESCRIBE-CAPABILITY), в третий operand (с возможностью только копирования, санкционированного в коде доступа) загружается потенциальный адрес того объекта (модуля, внутреннего или внешнего порта машины), с которым связана блокировка или ожидание. В противном случае в этот указатель помещается признак неопределенного значения.  
*Ошибки.* Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

*Примечание.* Для получения другой информации о процесс-машине следует воспользоваться командой DESCRIBE-CAPABILITY.

**Имя команды.** COMPUTE-PROCESS-MACHINE-CAPABILITY (CPMCAP)

**Выполняемая операция.** Вычисление значения потенциального адреса процесс-машины, выполняющей данную команду.

**Формат.** 0005, ОА

**Операнды.** Операнд должен быть указателем, в который помещается потенциальный адрес данной процесс-машины. В коде доступа санкционируется только копирование.

*Ошибки.* Основной набор ошибок, за исключением ошибок типа «несовместимые операнды» и «неопределенный operand».

**Имя команды.** CREATE-PORT

**Выполняемая операция.** Создание объекта «порт».

**Формат.** 0017, X, ОА

**Операнды.** Содержимое поля непосредственных данных (X) длиной 1 токен определяет, должен ли данный объект «порт» автоматически уничтожаться при разрушении процесс-машины, выполняющей данную команду. Значение xxx0 указывает на такую необходимость.

Операнд должен быть указателем. Он содержит потенциальный адрес порта, в коде доступа к которому санкционированы чтение, запись, уничтожение и копирование.

*Ошибки.* Основной набор ошибок (за исключением ошибок «несовместимые операнды» и «неопределенный operand»), а также ошибки типа «недостаточный объем памяти».

### Имя команды. SEND

**Выполняемая операция.** Передача значений указанных операндов (фактических параметров) через порт другому процессу. Выполнение команды не завершается до тех пор, пока другой процесс (по команде RECEIVE) не примет пересылаемые значения.

**Формат.** OB, OA, X, OA1,...,OAx

**Операнды.** Первый operand является указателем, адресующимся к определенному порту (в коде доступа указателя санкционирована запись). Содержимое поля непосредственных данных (X) длиной 2 токен определяет количество передаваемых фактических параметров (0—255); следующие за содержимым этого поля адреса operandов являются адресами параметров. Параметрами не могут быть литералы, срезы или содержимое ячеек «косвенный доступ к данным».

**Ошибки.** Основной набор ошибок, а также ошибки типа «неправильное число передаваемых данных». В случае несовпадения количества передаваемых (фактических) параметров с количеством принимающих (формальных) параметров в соответствующей команде RECEIVE регистрируется ошибка «неправильное число передаваемых данных». Если передаваемые (фактические) параметры несовместимы по типу с принимающими (формальными) параметрами при выполнении команды RECEIVE, то регистрируется ошибка «несовместимые operandы». Если значение какого-либо из фактических параметров не определено, регистрируется ошибка «неопределенный operand».

Любая ошибка, регистрируемая после начала пересылки данных («нарушение защиты доступа», если фактический параметр — указатель без санкционирования на копирование, «несовместимые operandы», «неопределенный operand»), влечет за собой завершение команды SEND и соответствующей ей команды RECEIVE с неполной передачей данных. Таким образом, эти ошибки, как и ошибка «неправильное число передаваемых данных», оказывают одновременное воздействие на команду SEND и соответствующую ей команду RECEIVE.

**Примечание.** Представление внешних устройств ввода-вывода (устройств без памяти) в качестве объектов «порт» системы и особенности использования команд SEND и RECEIVE в таких случаях рассматриваются в следующем разделе.

### Имя команды. RECEIVE

**Выполняемая операция.** Пересылка значений первого множества фактических параметров (каждое отдельное множество этих параметров является совокупностью фактических параметров, перечисляемых в одной команде SEND) заданного порта по указанным адресам operandов и изъятие их из порта. Если в

момент выполнения команды RECEIVE порт не содержит набора значений подобных параметров, выполнение этой команды не завершается до тех пор, пока группа соответствующих параметров не поступит в порт.

**Формат.** 0C, OA, X, OA1, ... OAх

**Операнды.** Первым операндом является указатель, адресующий к порту (в коде доступа к нему должно быть санкционировано чтение). В поле непосредственных данных (X) длиной 2 токен указывается число принимающих операидов. Следующие за содержимым этого поля адреса операидов определяют местоположение принимающих операндов. Последние могут быть операидами любого типа, но не литералами, срезами или содержимым ячейки «косвенный доступ к данным». (Для принимающих операндов не обязательно использовать ячейки «параметр», поскольку фактические параметры передаются в порт по значению. Если же принимающий операид — параметр, его значение пересыпается соответствующему фактическому параметру, переданному данному модулю при вызове последнего по команде CALL или LCALL.)

В отношении совместимости операидов SEND и RECEIVE справедливы те же правила, что и для команды ACTIVATE (атрибуты соответствующих посылающих и принимающих операидов команд SEND и RECEIVE должны быть одинаковы).

**Ошибки.** Основной набор ошибок, а также ошибки типа «неправильное число передаваемых данных». Если один или несколько фактических параметров имеют неопределенные значения, если число посылающих операидов (фактических параметров) в команде SEND не равно числу принимающих операидов (формальных параметров) в команде RECEIVE, если хотя бы один из операндов SEND не совместим с соответствующим операндом RECEIVE или если операндом SEND является указатель без санкции на копирование, то в обеих командах — RECEIVE и соответствующей команде SEND — регистрируются ошибки типа «неопределенный операид», «неправильное число передаваемых данных», «несовместимые операиды» или «нарушение защиты доступа».

### **Имя команды. GUARD**

**Выполняемая операция.** Если выполняемый модуль не находится в состоянии «охраняю», данная команда вводит его в это состояние и передает управление следующей команде. Если же выполняемый модуль пребывает в состоянии «охраняю», работа данной процесс-машины блокируется до момента выхода модуля из этого состояния.

**Формат.** 001С

**Ошибки.** Имитация ошибки.

**Примечания.** Если команда GUARD встречается после того, как ранее эта же процесс-машина ввела данный модуль в состояние «охраняем», то по этой команде не выполняется никаких действий. Единственной возможностью вывода модуля из состояния «охраняем» является выполнение команды UNGUARD. Выполнение команды RETURN или аномальное окончание данной активации модуля (например, вследствие возврата в программу обработки ошибок более высокого уровня) не оказывает влияния на состояние «охраняем» модуля.

#### **Имя команды.** UNGUARD

**Выполняемая операция.** Вывод модуля из состояния «охраняем».

**Формат.** 001D

**Ошибки.** Имитация ошибки.

#### **Имя команды.** DELAY

**Выполняемая операция.** Задержка работы процесс-машины на указанный интервал времени.

**Формат.** 0023, ОА

**Операнды.** Операнд должен быть целым числом. Если его значение отрицательное или равно нулю, то по команде DELAY не выполняется никаких действий. В противном случае работа процесс-машины приостанавливается на V мс, где V — значение операнда. В этом состоянии процесс-машины команды не выполняются, и на это время часть системных ресурсов, используемых процесс-машиной, может быть возвращена обратно системе (что именно подлежит возврату определяется конкретной реализацией вычислительной системы).

**Ошибки.** Основной набор ошибок, за исключением ошибок типа «несовместимые операнды».

### КОМАНДЫ ОТЛАДКИ

#### **Имя команды.** ENABLE

**Выполняемая операция.** Выполнение логической операции ИЛИ (OR) над содержимым заданного поля токенов и поля кода обрабатываемых ошибок в заголовке модуля с записью результата в поле этого кода. Результирующий код хранится в записи активации модуля, что означает, что данная команда воздействует только на текущую активацию модуля.

**Формат.** 0018, ОА

**Операнды.** Операнд должен быть полем токенов, размер которого (N) не превосходит длину поля кода обрабатываемых ошибок. Если указанное поле короче поля кода обрабатыва-

мых ошибок, в последнем могут быть изменены значения лишь первых N токен.

**Ошибки.** Основной набор ошибок (за исключением ошибок «несовместимые операнды»), а также ошибки типа «переполнение».

**Примечание.** Команды ENABLE и DISABLE не изменяют код обрабатываемых ошибок в самом модуле; их действия влияют только на текущую активацию модуля.

### **Имя команды.** DISABLE

**Выполняемая операция.** Логическое отрицание (подразрядная инверсия) содержимого заданного поля токенов с последующим выполнением логической операции И (AND) над этой величиной и кодом обрабатываемых ошибок в модуле и загрузкой результата в запись активации.

**Формат.** 0019, OA

**Операнды.** См. команду ENABLE

**Ошибки.** Основной набор ошибок (за исключением ошибок «несовместимые операнды»), а также ошибки типа «переполнение».

### **Имя команды.** RAISE-FAULT

**Выполняемая операция.** Формирование состояния «ошибка» с записью в поле непосредственных данных (X) длиной 2 токен величины, определяющей тип (номер) ошибки (т. е. значение, передаваемое программе обработки ошибок в качестве первого фактического параметра).

**Формат.** 000D, X

**Ошибки.** Возможны ошибки любого типа, задаваемого в поле непосредственных данных (X), а также ошибки «имитация ошибки». Значение X не должно быть нулевым или находиться в пределах 24—27. Если возникает такая ситуация, то регистрируется ошибка «недопустимая обработка ошибок». Если значение X не соответствует типам ошибок, предусмотренным архитектурой системы, ошибки считаются программно-определенными (28—255). Среди последних ошибки с четными номерами позволяют продолжение выполнения команд программы обработки ошибок после команды RAISE-FAULT, а при нечетных номерах ошибок этого не происходит.

### **Имя команды.** CONTINUE (CONT)

**Выполняемая операция.** Завершение выполнения программы обработки ошибок и передача управления команде, которая выполнялась бы следующей, если бы ошибка не была обнаружена.

**Формат.** 000E

**Ошибки.** Возможны ошибки типа «недопустимая обработка ошибок» [если в данный момент ошибки отсутствуют, если продолжение работы (по команде CONTINUE) для ошибки данного типа не допустимо или если команда CONTINUE принадлежит внутренней программе, вызванной программой обработки ошибок], а также ошибки типа «имитация ошибки».

**Примечание.** При необходимости выйти из программы обработки ошибок с целью повторения выполнения команды, вызвавшей ошибку, эта программа должна «потребовать» выполнения команды LRETURN. Если же необходимо перейти от обработки ошибки к выполнению команды, следующей за командой, вызвавшей ошибку, то программа обработки ошибок должна «потребовать» выполнения команды CONTINUE. Однако указанная команда может применяться только при регистрации ошибок типа «трассировка» или программируемых ошибок с четырьмя номерами (28—254), генерируемых командой RAISE-FAULT.

#### **Имя команды.** TRANSFER-FAULT (TRFAULT)

**Выполняемая операция.** Завершение выполнения данной программы обработки ошибок и вызов программы обработки ошибок более высокого уровня (более низкого в стеке записей активации). Если требуемая программа обработки не может быть найдена, выполнение текущего модуля завершается.

**Формат.** 0015

**Ошибки.** Возможны ошибки типа «недопустимая обработка ошибок» (в первом и третьем случаях, указанных в описании команды CONTINUE).

**Примечание.** Команда TRFAULT предназначена для использования в тех случаях, когда данная программа обработки ошибок предусматривает обработку ошибок встретившегося типа, однако после начала обработки принимается решение о ее обработке программой обработки ошибок более высокого уровня.

#### **Имя команды.** TRACE

**Выполняемая операция.** Разрешение или запрет на выполнение трассировки определенного типа для заданного модуля.

**Формат.** 0013, X, OA

**Операнды.** Первый операанд — содержимое поля непосредственных данных (X) длиной 2 токен — определяет тип трассировки: значение 0000xYxx задает трассировку переходов по условию «Да»; значение 0000xxYx — трассировку переходов по условию «Нет»; 0000xxxY — трассировку выполнения обращений по команде CALL; 0000Yxxx — трассировку маркера. Если Y=1, трассировка соответствующего типа разрешена; если Y=0, то запрещена. Второй операанд должен быть указателем, адресу-

ющимся к модулю с санкционированием записи в коде доступа. Задаваемое командой TRACE разрешение на выполнение трассировки того или иного типа сохраняет силу и для всех последующих активаций модуля.

Трассировка условных переходов по условиям «Да» и «Нет» может запрашиваться для всех команд типа «сравнения — переход» (COMPARISON-AND-BRANCH), за исключением команд ITERATE и ITERATE-REVERSE. Если реализация команды указанного класса приводит к необходимости выполнения перехода, а в данном модуле разрешена трассировка условных переходов по условию «Да», то регистрируется ошибка типа «трассировка». Аналогично если при реализации команды указанного класса возникает необходимость продолжения последовательного выполнения команд без перехода, а в модуле разрешена трассировка переходов по условию «Нет», то также регистрируется ошибка типа «трассировка». Если в модуле разрешена трассировка выполнения вызова по команде CALL, то ошибка «трассировка» регистрируется при каждом выполнении команды CALL, LCALL или CREATE-PROCESS-MACHINE. Если в модуле разрешена трассировка маркера, ошибка «трассировка» регистрируется при выполнении каждой команды MARKER. Обнаружение ошибки «трассировка» сопровождается передачей программе обработки ошибок в качестве пятого фактического параметра одного из следующих значений:

000001 — переход по условию «Да»;

000002 — переход по условию «Нет»;

000003 — выполнение команды CALL, LCALL или CREATE-PROCESS-MACHINE;

000004 — выполнение команды MARKER.

*Ошибки.* Основной набор ошибок, за исключением ошибок типа «неопределенный operand» и «несовместимые operandы».

*Примечание.* На систему возлагается функция обеспечения перехода на новый установленный режим трассировки не позднее, чем со следующего входа (например, вызовом по команде CALL, переходом в результате обработки ошибок или по команде RETURN) в указанный модуль.

### *Имя команды. MARKER*

*Выполняемая операция.* Генерирование ошибки типа «трассировка», если разрешена трассировка маркера. Если такого разрешения нет, то никаких действий по данной команде не выполняется.

### *Формат. F*

*Ошибки.* Возможны ошибки типа «трассировка» и «имитация ошибки».

### Имя команды. RANGE-CHECK (RANGECHK)

Выполняемая операция. Регистрация ошибки «выход за пределы допустимых значений», если значение первого операнда меньше значения второго операнда или больше значения третьего операнда.

Формат. OF, OA, OA, OA

Операнды. Операнды должны быть числами или символами. Если это числа, то они должны быть одного и того же типа (целыми, целыми увеличенной разрядности, порядковыми, десятичными с фиксированной или плавающей точкой). Если учитывать это требование, то данная команда оказывается эквивалентной выполнению последовательности команд:

```
GEBF OAI,OA2,!1  
LEBF OAI,OA3,!1
```

где !1 обозначает адрес команды RAISE-FAULT, определяющей ошибку «выход за пределы допустимых значений».

Ошибки. Основной набор ошибок, а также ошибки типа «выход за пределы допустимых значений».

## ОСОБЕННОСТИ ДЕЙСТВУЮЩЕЙ СИСТЕМЫ РАССМАТРИВАЕМОЙ АРХИТЕКТУРЫ

При реализации архитектуры системы SWARD на практике особое внимание следует уделить трем командам, от которых зависит надежность функционирования механизма потенциальной адресации. Команда CREATE-MODULE должна сканировать адресное пространство и область размещения команд с целью проверки, все ли адреса операндов, являющиеся адресами ячеек, представляют собой обращения ко всей ячейке. Не исключаются случаи создания программ, команды которых содержат обращения к отдельным элементам ячеек, что, вообще говоря, можно рассматривать как не предусмотренные архитектурой возможности создания потенциальных адресов. Это можно предотвратить посредством команды CREATE-MODULE, сканирующей сначала адресное пространство с целью определения начала каждой ячейки, а затем прибегнув к другой команде для выяснения, являются ли адреса ячеек допустимы для использования.

В качестве альтернативного решения данной проблемы можно было бы предложить использовать в качестве адресов операндов *номера ячеек*, т. е. ввести нумерацию ячеек: ячейка 1, ячейка 2 и т. д. Однако при этом вводился бы дополнительный уровень адресации при обработке команды. (Можно было бы пользоваться номерами ячеек во внешнем модуле, а посредством команды CREATE-MODULE преобразовывать их в адреса ячеек объекта «модуль», но это не проще сканирования, кото-

рое должна выполнять эта команда для определения, допустимы ли адреса ячеек к использованию.)

Сказанное относится и к командам COMPUTE-CAPABILITY-EXTERNALLY и COMPUTE-ENTRY-CAPABILITY. Вычислительная система должна удостовериться, что она адресуется ко всей ячейке полностью, а команды, к которым производится обращение, содержат адреса ячеек, допустимые к использованию.

Ни одна из команд системы не является прерываемой, т. е. командой, имеющей определенные промежуточные состояния, позволяющие приостановить выполнение команды, обработать другие команды, а затем продолжить выполнение прерванной команды. Некоторые вычислительные системы имеют прерываемые команды (например, команда MOVE-CHARACTER-LONG в Системе 370). В системе SWARD от использования таких команд отказались по двум причинам. Во-первых, почти все команды системы SWARD могут выполняться достаточно долго; предусмотреть возможность их прерывания означало бы необычайное усложнение архитектуры системы. Во-вторых, принцип прерываемости команд не соответствует заложенным в данную архитектуру принципам функционирования процесс-машин. Если придерживаться этих принципов, то при сравнительно низкой стоимостности в настоящее время реальных процессоров, естественным оказывается построение системы SWARD на нескольких процессорах (конечно, не следует отождествлять их количество с количеством процесс-машин, поскольку последние могут свободно создаваться и уничтожаться). Если же создать реальную систему, в которой несколько недорогих процессоров поддерживают существование переменного числа процесс-машин, необходимость в прерываемых командах отпадает.

Отметим, что в описаниях команд DESCRIBE-CAPABILITY, CONTROL-PROCESS-MACHINE и CREATE-PROCESS-MACHINE используется понятие приоритетов процесс-машин, не детализируемое архитектурой и уточняемое на этапе реализации системы. Так, на основе различия приоритетов процесс-машин можно организовать очередность их доступа к памяти, определить объем работ, выполняемых для каждой процесс-машиной небольшим набором реальных процессоров, управлять местоположением каждой процесс-машиной во внешних очередях (например, у портов или в ожидании выхода модуля из состояния «охраняется», задаваемого командой GUARD).

В рассматриваемой системе запись активации является в определении смысле уникальным объектом, поскольку допускает потенциальную адресацию только к ячейке внутри объекта, но не ко всему объекту. Потенциальный адрес будет относиться к

ячейке внутри записи активации, если он рассчитывается модулем для ячейки, относящейся к динамической части адресного пространства. Поскольку это встречается редко и, кроме того, в связи с тем что записи активации являются наиболее часто создаваемыми объектами (большинство системных имен приходится на них), целесообразно формировать для них уникальные имена автоматически. Предпочтение следует отдать такому решению, при котором команды, вычисляющие потенциальные адреса, например команда COMPUTE-CAPABILITY, при обращении к ячейкам из динамической части адресного пространства определяют, имеет ли уже запись активации системное имя (SON). Если имени нет (первое обращение к области), то оно присваивается во время выполнения команды.

#### ОСОБЕННОСТИ ПРЕДОСТАВЛЕНИЯ ПАМЯТИ ПРОЦЕСС-МАШИНАМ

Система SWARD является моделью вычислительной машины, имеющей неограниченный резерв процессоров (процесс-машин) при фиксированном, хотя и большом, объеме памяти, являющейся памятью одного уровня. «Порциями» фиксированного объема эта память может распределяться между всеми существующими в данный момент процесс-машинами.

Как было упомянуто выше, предусмотрена специальная единица измерения объема памяти — АМ (amount of storage — порция памяти). Конкретное значение АМ не определяется архитектурой системы SWARD; это значение задается в процессе практической реализации вычислительной системы.

При реализации системы SWARD допускается также задание некоторого минимального значения АМ для любого объекта, размер которого меньше «стандартного», т. е. фиксированного для проектируемого варианта системы. Это делается с целью экономии ресурсов системы, «поглощаемых» процесс-машиной, которая (согласно принципам архитектуры системы SWARD) должна помнить все имена существующих в системе объектов.

Количество АМ, выделяемое объектам конкретной реализованной системы, можно узнать, предусмотрев вывод на печать таблицы израсходованных АМ. Подобную информацию можно получить также путем создания интересующих пользователя объектов с последующим выполнением команды DESCRIBE-CAPABILITY для определения количества АМ, выделенного этим объектам.

Для существующей реализации системы SWARD можно составить представление о величине АМ на основании следующих примеров:

- 1) 200 объектов «порт» занимают  $\sim 1$  АМ;
- 2) объекту «память данных» для массива из 200 10-символьных полей необходим  $\sim 1$  АМ;
- 3) объект «модуль», соответствующий изображенному на рис. 14.11 внешнему модулю, занимает 0,1 АМ;
- 4) записи активации модуля «объект» требуется  $<0,1$  АМ;
- 5) объекту «процесс-машина» необходим 0,1 АМ;
- 6) процесс, не создающий явным образом новых объектов и формирующий сравнительно мало записей активации с небольшим количеством ячеек, соответствующих динамической части адресного пространства, требует для своего выполнения  $\leq 1$  АМ.

При конкретной реализации архитектуры системы SWARD допускается также организация подсистем так называемого *быстрого распределения памяти* для ограниченного числа небольших, часто создаваемых объектов (например, записей активации, длина которых меньше некоторой пороговой величины). Быстрое распределение памяти производится без обращения к памяти процесс-машины (без уменьшения величины АМ имеющейся в ее распоряжении памяти). При этом при выдаче информации о памяти, занимаемой объектами, требующими менее 1 АМ памяти, система может генерировать нулевое значение.

Количество требуемых единиц АМ можно рассматривать как характеристику (атрибут) объекта. В то же время для объекта «процесс-машина» можно указать еще одну характеристику: *количество имеющихся в наличии АМ*, т. е. объем имеющейся в распоряжении свободной памяти. Речь идет о памяти, которой располагает данная процесс-машина для создания новых объектов. Имеющаяся свободная память перераспределяется между процесс-машинами как во время их создания, так и при их уничтожении. Она может также перераспределяться динамически по специальному запросу. Когда создается новая процесс-машина, ей передается определенная часть свободной памяти, которой располагает исходная процесс-машина. Переданная процесс-машине свободная память становится свободной памятью этой машины. Величина переданной свободной памяти вычитается из величины АМ свободной памяти, имеющейся в исходной процесс-машине. Перераспределение свободной памяти между данной процесс-машиной и другой процесс-машиной или между двумя другими процесс-машинами может выполняться с помощью команды TRANSAM.

При начальном запуске система содержит одну процесс-машину, выполняющую программу определенного встроенного модуля. Эта процесс-машина охватывает всю свободную память, имеющуюся в системе.

При создании любого нового объекта (в том числе объекта «процесс-машина») объем свободной памяти, имеющейся в распоряжении исходной процесс-машины, уменьшается на величину (измеряемую в АМ), которая необходима для создания объекта. Если количества АМ свободной памяти для этой цели недостаточно, объект не создается и регистрируется ошибка «недостаточный объем памяти».

Когда некоторый объект уничтожается по явному запросу, размер занимаемой им памяти (в АМ) добавляется к количеству АМ свободной памяти процесс-машины, выполняющей команду DESTROY. Если уничтожаемым объектом является процесс-машина, то к количеству АМ свободной памяти процесс-машины, выполняющей команду DESTROY, добавляется также количество АМ свободной памяти уничтожаемой процесс-машины.

При уничтожении некоторого объекта косвенным путем (как следствие уничтожения процесс-машины) занимаемая им память добавляется к свободной памяти уничтожаемой процесс-машины<sup>1)</sup>. Отметим, что процесс-машина может косвенным путем уничтожить саму себя (при возврате управления из первоначальной активации), а также может уничтожить себя по явному запросу командой DESTROY. В этих случаях свободная память этой процесс-машины передается процесс-машине (если она еще существует), обусловившей появление данной процесс-машины, или процесс-машине, инициировавшей этот процесс.

Все упомянутые случаи перераспределения памяти, выполняемые системой, можно реализовать по явному запросу с помощью команды TRANSAM.

Отметим, что понятие процесс-машины является независимым (ортогональным) по отношению к другим понятиям, связанным с архитектурой системы SWARD. Например, нет взаимозависимости функционирования процесс-машин и системы адресации.

## ЭФФЕКТИВНОСТЬ ВЫПОЛНЕНИЯ СИСТЕМОЙ SWARD ОПЕРАТОРОВ ЯЗЫКОВ ВЫСОКОГО УРОВНЯ

Анализировать эффективность работы SWARD можно по крайней мере по трем основным направлениям. Одно из них — анализ «на макроуровне». В этом случае интересуются не конкретным быстродействием машины, а общей эффективностью решения основных задач, стоящих перед системой (т. е. эффек-

<sup>1)</sup> В конечном счете — к количеству АМ свободной памяти процесс-машины, вызвавшей уничтожение. — Прим. перев.

тивностью обеспечения интерфейса человек — машина). Учитывая общую ориентацию архитектуры системы SWARD на создание благоприятной среды для разработки и выполнения программ, нельзя считать ие имеющим под собой основу утверждение, что при таком подходе система SWARD отличается намного более высокой эффективностью по сравнению даже с современными сверхбольшими ЭВМ. Поскольку, одиако, для оценки подобной эффективности трудно предложить определенную меру, рассматриваемый подход ие позволяет дать количественную оценку системы SWARD.

Второе возможное направление анализа эффективности системы SWARD предполагает рассмотрение «на уровне системы». В этом случае объектом оценки также ие является время выполнения операторов, им служит эффективность выполнения стандартных системных программ. Рассматривается влияние особенностей архитектуры на реализацию различных функций системы: влияние фундаментальных принципов системы (потенциальной адресации, распределения вычислительного процесса между отдельными процесс-машинами, пересылки данных через порты) на работу программ операционной системы, влияние повышенной компактности программ на пересылку данных между запоминающими средами различного уровня.

Третье направление анализа эффективности системы SWARD связано с использованием традиционных оценок работы системы на «микроуровне». Здесь критерием эффективности считается скорость выполнения операторов небольших фрагментов программ. На первый взгляд может показаться, что по этому критерию система SWARD имеет плохие показатели. Однако это ие так, и главным образом благодаря большому значению параметра (меры)  $M$ .

В качестве примера рассмотрим эффективность обработки строк символов различными вычислительными системами. В частности проанализируем операции выделения подстроки, вычисления ее длины, поиска подстроки и ее присоединения к строке на языке ПЛ/1 для строк постоянной и переменной длины, а также для строк, длина которых определяется в момент вызова процедуры (использование символа \* для указания длины в определениях строк на языке ПЛ/1). Сравнение проводится с моделью 158 Системы 370, в которой используется разработанный фирмой IBM оптимизирующий компилятор языка ПЛ/1.

Текст рассматриваемой программы на языке ПЛ/1 представлен на рис. 15.1. При выполнении процедуры передавались параметры следующих типов: CHARACTER(80) VARYING; CHARACTER(6); FIXED-BINARY(15) со значением 1; FIXED-BINARY(15) со значением 5.

```

STRTEST: PROCEDURE (A,B,I,J); /* STRINGSIZE,STRINGRANGE OFF */
DCL A CHAR(*) VARYING;
DCL B CHAR(*);
DCL C CHAR(20) VARYING;
DCL D CHAR(8) INIT('12345678');
DCL E CHAR(8);
DCL F CHAR(1);
DCL G CHAR(2) INIT('DE');
DCL H CHAR(80);
DCL X BIT(4);
DCL Y BIT(16) VARYING INIT('1010010001000010'B);
DCL (I,J,K,L,M) FIXED BINARY(15);
130 C = SUBSTR(A,I,J); /* C = 'AAAAAA' LENGTH 5 */
/* SUBSTR TO LOCAL VARYING FROM PARAM * VARYING */
150 F = SUBSTR(D,J,I); /* F = '5' */
/* SUBSTR TO LOCAL FROM LOCAL */
170 SUBSTR(E,I,I+J) = B; /* E = 'ABCDEF???' */
/* SUBSTR INTO LOCAL FROM PARAM */
190 SUBSTR(E,I,J) = SUBSTR(C,I,J); /* E = 'AAAAAAF???' */
/* SUBSTR INTO LOCAL FROM SUBSTR FROM LOCAL VARY */
210 C = C || G; /* C = 'AAAAAADE' LEN 7 */
/* CONCAT LOCAL TO LOCAL VARYING */
230 C = C || B; /* C = 'AAAAAADEABCDE' LEN 13 */
/* CONCAT PARAM * TO LOCAL */
250 K = LENGTH(A); /* K = 80 */
/* LENGTH OF PARAM * VARYING */
270 M = LENGTH(C); /* M = 13 */
/* LENGTH OF LOCAL VARYING */
290 H = A; /* H = AAAAAAAAAABBB... (80 CHARS) */
/* 80-CHAR MOVE FROM PARAM * VARYING */
310 L = INDEX(A,G); /* L = 40 */
/* INDEX OF LOCAL 2-CHAR IN PARAM * VARYING */
330 I = INDEX(H,B); /* I = 0 (NOT FOUND) */
/* INDEX OF PARAM * IN LOCAL */
350 X = SUBSTR(Y,J,4); /* X = '0100'B */
/* SUBSTR TO LOCAL BIT FROM LOCAL BIT VARYING */
END STRTEST;

```

Рис. 15.1. Программа на языке ПЛ/1.

Текст программы не требует дополнительных пояснений. Все операторы, кроме оператора, находящегося в строке с номером 350, выполняют стандартные функции по обработке строк символов. Что касается операции выделения подстроки в строке битов, она встречается относительно редко, поскольку ее реализация в Системе 370 связана с преодолением ряда затруднений. С целью упрощения сопоставления результатов работы сравниваемых систем упомянутую операцию можно не принимать в расчет.

Для целей сравнения использовались два варианта программы на языке ПЛ/1. Один из них, как было упомянуто выше, представлен на рис. 15.1. Второй вариант программы отлича-

ется от первого только наличием программных средств STRING SIZE и STRINGRANGE, обеспечивающих на этапе выполнения программы проверку двух условий: принадлежности адреса подстроки области, занимаемой строкой, и отсутствия переполнения при выполнении присваивания значения строке символов. Имея в виду защищенность программы от ошибок программиста, первый из двух рассматриваемых вариантов можно назвать «незащищенной» программой, а второй — «защищенной». При создании программ для производственной эксплуатации безусловно целесообразно включение в их текст средств проверки

```

STRTEST MODULE
  DATA SPACE
    A DCL CHAR,(*,VARYING),PARAMETER
    B DCL CHAR,(*) ,PARAMETER
    C DCL CHAR,(20,VARYING),AUTOMATIC
    D DCL CHAR,(8),INIT='12345678',AUTOMATIC
    E DCL CHAR,(8),AUTOMATIC
    F DCL CHAR,(1),AUTOMATIC
    G DCL CHAR,(2),INIT='DE',AUTOMATIC
    H DCL CHAR,(80),AUTOMATIC
    X DCL BOOLEAN,(4),AUTOMATIC
    Y DCL BOOLEAN,(16,VARYING),INIT=B'1010010001000010',AUTOMATIC
    I DCL INTEGER,PARAMETER
    J DCL INTEGER,PARAMETER
    K DCL INTEGER,AUTOMATIC
    L DCL INTEGER,AUTOMATIC
    M DCL INTEGER,AUTOMATIC
    T1 DCL INTEGER,AUTOMATIC
    T2 DCL INTEGER,AUTOMATIC
    INSTRUCTION SPACE
      ACT      (A,B,I,J).
      MOVESS C,I,J,A,I,J
      MOVESS F,I,I,D,J,I
      MOVE     T1,I
      ADD     T1,J
      LENGTH T2,B
      MOVESS E,I,T1,B,1,T2
      MOVESS E,I,J,C,I,J
      CCAT     C,G
      CCAT     C,B
      LENGTH K,A
      LENGTH M,C
      MOVE     H,A
      MOVE     L,I
      INDEX   L,0,A,G
      MOVE     I,1
      INDEX   I,0,H,B
      MOVESS X,I,4,Y,J,4
      RETURN
  STRTEST MODULE END
END

```

Рис. 15.2. Программа на языке ассемблера системы SWARD.

выполнения указанных выше условий (выявление возможностей возникновения ошибок). Поэтому при оценке эффективности выполнения программ следует сравнивать именно «защищенную» программу с эквивалентной ей программой системы SWARD. При этом, конечно, благодаря обширному семантическому контролю, выполняемому системой SWARD, ее программа менее подвержена ошибкам, чем «защищенная» программа на языке ПЛ/1.

Остановимся на некоторых событиях, которые имели место при работе с рассматриваемой процедурой на языке ПЛ/1. При написании внешней процедуры, которая должна передавать фактические параметры процедуре, текст которой приведен на рис. 15.1, третий и четвертый из передаваемых параметров во внешней процедуре были представлены не 15-разрядными двоичными числами (как это необходимо), а 31-разрядными. Выполнение программы протекало «успешно» при использовании случайных значений I и J, и примерно целый день проводился анализ листинга трассировки, полученного при использовании модели 158, прежде чем случайно была обнаружена эта ошибка. Система SWARD такую ошибку выявила сразу. Более того, когда ошибка была исправлена и программа выполнялась системой SWARD в режиме трассировки, последняя обнаружила ошибку «несовместимые операнды» при выполнении команды ACTIVATE в начале этой процедуры. Эта

Таблица 15.4. Статистические данные о количестве выполняемых команд

Номер оператора программы	Количество команд при		
	«незащищенным» варианте программы на языке ПЛ/1 для Системы 370	«защищенным» варианте программы на языке ПЛ/1 для Системы 370	использовании программы для системы SWARD
Вызов/Возврат			
130	53	142	3
150	21	129	1
170	12	119	1
190	26	123	4
210	33	196	1
230	26	89	1
250	30	93	1
270	3	3	1
290	2	2	1
310	14	66	1
330	153	153	2
350	364	364	2
	121	168	1
Всего	858	1647	20

ошибка означает несоответствие типов передаваемых и принимаемых данных. Оказалось, что в вызывающей процедуре первый параметр при обращении к данной процедуре не был объявлен как VARYING. Из этого следует, что во время работы Системы 370 в ряде случаев в операнде не досчитывалось двух символов. Однако ошибка не была обнаружена до тех пор, пока программа не была выполнена системой SWARD.

Эквивалентная программа на языке ассемблера системы SWARD представлена на рис. 15.2.

В табл. 15.4 приведены статистические данные о выполняемых командах. Первая строка таблицы содержит количество команд, выполняемых для вызова процедуры. К указанным командам относятся следующие: 1) последовательность команд внешней процедуры, обеспечивающих вызов; 2) команды пролога в вызываемой процедуре и 3) команды возврата управления в вызывающую процедуру.

В табл. 15.5 приведен один из важных показателей особенностей реализации быстродействия системы, который не зависит от конкретной реализации системы. В ней указано количество

Таблица 15.5. Объем пересылаемых данных для разных программ

Номер оператора программы	Объем данных (в байтах) при		
	«незашщщенному» варианту программы на языке ПЛ/1 для Системы 370	«зашщщенному» варианту программы на языке ПЛ/1 для Системы 370	использовании программы для системы SWARD
Вызов/Возврат			
·130	506	1132	711 <sup>1)</sup>
·150	134	936	86,5
·170	70	858	68
·190	142	874	153
·210	188	1349	89,5
·230	144	649	25
·250	182	709	39
·270	26	26	25,5
·290	12	12	19,5
·310	229	614	185,5
·330	880	880	131,5
·350	1668	1668	288
Прочитано байтов	867	1040	44,5
Записано байтов	4461	8960	1417,5
	587	1787	449 <sup>1)</sup>
Всего	5048	10 747	1866,5

<sup>1)</sup> Большие значения этих двух чисел определяются тем, что в системе SWARD всем локальным переменным присваиваются неопределенные значения (например, переменной Н присваивается 80 символов неопределенного значения).

ство байтов данных, пересылаемых между процессором и памятью при выполнении программы (так называемая мера M). Отметим, что регистры общего назначения Системы 370 считаются частью процессора. В соответствии с этим по команде LOAD-REGISTER пересыпаются 2 байт данных, а по команде LOAD — 8 байт (4 байт для команды и 4 байт для извлекаемого из памяти 32-разрядного двоичного слова).

Информация, относящаяся к Системе 370, хотя и получена в результате трассировки выполнения программы на модели 158, учитывает только те байты данных, которые необходимы в соответствии с принципами, заложенными в архитектуру системы. Так, при использовании модели часто поступает запрос в память на 4 или 8 байт данных, когда в действительности требуемая информация занимает меньший объем памяти. В этой системе выполняется опережающая выборка следующей по порядку команды, несмотря на то что в программе может быть нарушено последовательное выполнение команд и осуществлена передача управления. Пересылка этой дополнительной информации, определяемой спецификой реализации архитектуры системы, не учитывается в количественных показателях Системы 370, приведенных в табл. 15.5. Однако в показателях системы SWARD этот параметр учитывается. Например, процессор системы SWARD может извлекать из памяти 6 токен (3 байт), хотя в действительности ему необходима только часть этих данных. Таким образом, сведения о Системе 370 содержат информацию, минимально необходимую, согласно принципам архитектуры этой вычислительной системы, а аналогичные сведения о системе SWARD носят несколько завышенный характер.

В табл. 15.6 приведено время выполнения операторов при использовании модели 158, базовой модели системы SWARD и реально не существующей машины SWARD 158, описываемой ниже. Количественные показатели для модели 158 получены на основе приводимых в документации характеристик системы.

При сопоставлении количественных показателей систем следует принять во внимание их зависимость от особенностей реализации архитектуры. На первый взгляд модель 158 представляется более быстродействующей, чем базовая модель системы SWARD. Однако необходимо учесть ряд факторов. Так, система SWARD располагает процессором небольшой мощности (порядка 9000 вентилей с представлением данных только 24 бит, а не 32 бит, как в модели 158). Модель 158 имеет более высокое быстродействие памяти ( $24 \cdot 10^6$  байт/с для кэш-памяти,  $13 \cdot 10^6$  байт/с для основной памяти; тот же параметр для базовой модели системы SWARD составляет  $10 \cdot 10^6$  байт/с). У модели 158 более короткий основной цикл (115 нс

**Таблица 15.6.** Время выполнения операторов (значения округлены с точностью до микросекунды)

Номер опе- ратора про- граммы	Время выполнения операторов (в микросекундах) при			
	«незащищенным» варианте программы на языке ПЛ/1 для модели 158	«защищенным» варианте про- грамм на язы- ке ПЛ/1 для модели 158	использовании программы для базовой модели SWARD	использования программы для модели SWARD 158
Вызов/Воз- врат				
130	63	147	98	79
150	18	118	24	18
170	10	109	19	15
190	22	110	35	28
210	26	170	22	17
230	21	86	9	7
250	24	93	10	8
270	3	3	6	5
290	250	2	5	4
310	24	70	23	19
330	204	204	48	39
350	396	396	91	72
	122	146	18	14
Всего	935	1654	409	325
Нормализо- ванное значение (по отно- шению к «незащи- щенному» варианту програм- мы на языке ПЛ/1)	1,00	1,77	0,44	0,35
Нормализо- ванное значение (по отно- шению к «защи- щенному» варианту програм- мы на языке ПЛ/1)		1,00	0,25	0,20

по сравнению со 160 ис в системе SWARD). Система SWARD выполняет намного больше семантических проверок (например, проверок типа данных), чем даже «защищенный» вариант программы на языке ПЛ/1. Кроме того, в процессоре 158 имеется буфер команд, позволяющий осуществлять опережающую выборку нескольких команд. Однако ни в одной из этих машин не используется принцип конвейерной обработки команд.

При сопоставлении быстродействия различных систем следует, конечно, обращать внимание на быстродействие обмена с памятью (т. е. на ее пропускную способность). Пропускная способность процессора определяется относительно просто (цикл обращения к памяти составляет 300 ис, отсутствует раслоение и за каждый цикл может быть записано или прочитано до 3 байт данных). Аналогичные показатели для модели 158 зависят от типа доступа к памяти и от коэффициента совпадения для кэш-памяти. Показатели пропускной способности памяти для модели 158 Системы 370 представлены в табл. 15.7.

Чтобы при сопоставлении систем различной архитектуры по возможности исключить влияние особенностей конкретных реализаций, была теоретически рассмотрена модель системы SWARD с процессором, которому присуща специфика реального процессора 158. В целом это тот же процессор системы SWARD (без буфера команд, с 24-битовым представлением данных), однако его основной цикл и скорость обмена с памятью такие же, как у процессора модели 158. Расчеты были выполнены посредством пакета имитационного моделирования программного обеспечения системы SWARD. Пакет позволяет задавать скорости работы процессора и обмена с памятью в качестве параметров, однако не допускает имитировать кэш-память. Но поскольку кэш-память в первом приближении можно рассматривать как средство повышения скорости обмена, в модель за-

Таблица 15.7. Пропускная способность памяти для модели 158 Системы 370

Форма обмена	Максимальная скорость пересылки данных, Мбайт/с
1. Чтение последовательно расположенных слов из кэш-памяти	23,8
2. Чтение случайным образом расположенных слов из кэш-памяти	17,4
3. Чтение последовательно расположенных слов из основной памяти	12,6
4. Чтение случайным образом расположенных слов из основной памяти	3,9
5. Запись двойных слов (по 8 байт) в основную память	11,6
6. Запись слов (по 4 байт) в основную память	4,4

кладывалось примерно среднее значение скорости обмена с памятью для модели 158 (с учетом повышения быстродействия за счет кэш-памяти).

Данные, полученные для системы SWARD 158, могут быть также использованы для определения того, насколько показатель  $M$  характеризует скорость выполнения операций. Значения  $M$  были следующими: 10 747 — для «зашщищенной» программы на языке ПЛ/1 и 1866,5 — для системы SWARD. Время выполнения соответствующей программы машиной 158 составило 1654 мкс. Используя значение  $M$ , можно высказать предположение, что для системы SWARD 158 время выполнения оказалось бы равным 287 мкс. В результате же расчетов методом имитационного моделирования была получена величина, равная 325 мкс. Расхождение можно считать незначительным, если принять во внимание, что процессор системы SWARD 158 не обладает всеми функциональными возможностями мощного процессора модели 158. Для «незашщищенного» варианта программы на языке ПЛ/1 величина  $M$  оказалась равной 5048 (при 1866,5 для системы SWARD). Исходя из времени выполнения этой программы на модели 158, равного 935 мкс, соответствующее время для системы SWARD оценивается в 345 мкс, что также незначительно отличается от значения, равного 325 мкс, полученного путем моделирования.

Наконец, в табл. 15.8 приводятся размеры загрузочных модулей рассматриваемой программы. Отметим, что для программы системы SWARD учитывается память, занимаемая как командами, так и тегами.

В табл. 15.9 дано время выполнения основных команд и типичных операций. В общем случае оценить время выполнения оказывается нелегко по следующим причинам: 1) это время зависит от способа адресации операидов (в частности, от того, является ли операид параметром или косвенным доступом к нему); 2) вследствие оптимизации микропрограмм, реализующих многие команды, скорость выполнения последних оказывается

Таблица 15.8. Размер памяти, занимаемой разными программами

Вариант программы	Размер загрузочного модуля, байт
«Незашщищенный» вариант программы на языке ПЛ/1 для Системы 370	970
«Зашщищенный» вариант программы на языке ПЛ/1 для Системы 370	1340
Программа для системы SWARD (включая память, занимаемую тегами)	135,5

Таблица 15.9. Время выполнения некоторых команд и операций

Команда (операция)	Время, мкс
BRANCH	0,5
MARKER	0,5
MOVE (литерал в ячейку «целое число»)	3,6
MOVE [литерал в целочисленный элемент массива A (I)]	8,5
MOVE (целое число, включающее 50 элементов массива или «реза»)	25,5
ADD (литерал в ячейку «целое число»)	3,9
MULTIPLY (целые числа)	6,8—10,5
RANGECHECK (целое число, литерал, целое число)	6,0
ITERATE	3,7
SUBTRACT (7-разрядные десятичные числа)	5,4
ALLOCATE (для 80-символьной строки)	15,5
CREATE PROCESS MACHINE	39,8
SEARCH (массив или срез целых чисел)	0,6 на элемент
SEND (один фактический параметр в пустой порт)	21,8
CREATE PORT	12,2
CONVERT (целое число, включающее 8 символов)	19,1
CONVERT (<—45.78 — число с фиксированной точкой)	21,9
CALL	25—40 <sup>1)</sup>
ACTIVATE (5 параметров)	13,5
RETURN	7,5
Переключение процессов	7,7

<sup>1)</sup> Диапазон типичных значений, которые в значительной степени зависят от размера создаваемой записи активации.

зависимой от типа обрабатываемых данных; 3) время выполнения отдельных команд зависит от сложившихся в программе к данному моменту условий (так, например, обстоит дело с командами, которые для создания объектов предварительно запрашивают память).

Если ввести некоторые усовершенствования, можно заметно уменьшить время выполнения многих команд. Например, была разработана, хотя и не включена в базовую модель системы SWARD, подсистема памяти с сохранением информации о последних выполненных операциях. Она представляет собой небольшую ассоциативную память, содержащую восемь адресов ячеек, к которым производилось обращение в последних выполнившихся командах. Для каждого из этих адресов хранятся данные о первых 6 токен тега ячейки и физическом адресе ее текущего содержимого. Моделирование этой небольшой дополнительной подсистемы показало, что она может уменьшить время выполнения основных команд из числа перечисленных в табл. 15.9 (пересылка скалярных величин, сложение, сравнение и т. п.) на 30—50%.

---

## ЧАСТЬ VI

# МИКРОПРОЦЕССОР С АРХИТЕКТУРОЙ, ОРИЕНТИРОВАННОЙ НА ОБЪЕКТЫ

### ГЛАВА 16

#### ОСНОВНЫЕ ПРИНЦИПЫ РАБОТЫ МИКРОПРОЦЕССОРА iAPX 432

За последние пять-десять лет наиболее значительным достижением на пути создания новых типов архитектуры ЭВМ следует считать разработку фирмой Intel микропроцессора iAPX 432. В основу этой микропроцессорной системы положены многие принципы и понятия, рассмотренные в гл. 4 (такие, как «потенциальная адресация», «объекты», «области санкционированного доступа», «средства управления процессами» и т. п.). Налажен массовый выпуск упомянутых типов микропроцессоров, что свидетельствует о практической пригодности упомянутых принципов и понятий. Реализация микропроцессорной системы iAPX 432 является значительным достижением в области конструирования систем на больших интегральных схемах, поскольку ее процессор выполнен на двух интегральных схемах, содержащих по 160 000 полупроводниковых элементов. Только на разработку кремниевой подложки и внутренней топологии элементов систем затрачено 100 человеко-лет.

Поскольку архитектура системы iAPX 432 имеет много общего с архитектурой системы SWARD, по мере возможности будем проводить сравнение этих систем. Предполагается, что читатель ознакомился с гл. 13—15. Для удобства сравнения рассмотрение системы iAPX 432 будет проводиться примерно в том же порядке, что и описание системы SWARD.

#### ЦЕЛИ СОЗДАНИЯ АРХИТЕКТУРЫ СИСТЕМЫ iAPX 432

Главная задача, стоявшая перед разработчиками системы iAPX 432, заключалась в существенном сокращении стоимости и времени разработки прикладных программ для микропроцессорной системы. Достижение поставленной цели оказалось возможным благодаря реализации пяти из шести основных требований, которым, как указано в гл. 13, должны удовлетворять

эффективные вычислительные системы. (Первое из требований — обнаружение семантических ошибок в программах — как цель проектирования системы iAPX 432 не выдвигалось.) Так, ограничение последствий возможных ошибок в программном обеспечении системы iAPX 432 достигается благодаря выбору соответствующих принципов адресации и защиты от несанкционированного доступа. Архитектура iAPX 432 такова, что облегчается процесс проектирования программного обеспечения на основе прогрессивных технологических приемов структурирования, программы для системы iAPX 432 могут быть приведены к более простому виду, чем аналогичные программы для других систем. Имеются средства для эффективности тестирования и отладки программ; программы операционной системы и другие подобные средства программного обеспечения отличаются сравнительной простотой.

Основная цель разработки архитектуры системы iAPX 432 — усовершенствование процесса разработки прикладного и системного программного обеспечения — обусловлена тем обстоятельством, что именно трудности разработки программ являются основным препятствием для широкого внедрения вычислительной техники. Достижение поставленной цели проектирования микропроцессора iAPX 432 предопределяет и область его наиболее широкого применения — создание систем, отличающихся относительно большой долей стоимости разработки программного обеспечения в общей стоимости системы. Маловероятно, что микропроцессор iAPX 432 будет использоваться во многих традиционных областях применения микропроцессоров, например в системах управления и контроля функционирования электроприводов, автомобилей или терминалов. Это связано с тем, что в подобных специализированных вычислительных системах программирование, как правило, выполняется однократно, поэтому его стоимость составляет малую долю стоимости системы. Скорее всего микропроцессор iAPX 432 найдет применение в следующих случаях:

- 1) в более простых объектах, для которых, однако, требуется достаточно большой объем программирования;
- 2) в сложных системах, нуждающихся в иеоднократном перепрограммировании;
- 3) в системах, функционирование которых предполагает наличие ряда одновременно выполняющихся параллельных процессов (организации параллельного выполнения процессов в микропроцессоре iAPX 432 уделяется большое внимание);
- 4) в системах, для которых необходимо обеспечить надежную защиту от ошибок программирования (эта характеристика программных средств является еще одной важной особенностью микропроцессора iAPX 432).

Вторая цель создания системы iAPX 432 состоит в наделении проектируемых программных средств свойством адаптируемости структуры, т. е. в обеспечении возможности динамического добавления или удаления процессоров, обслуживающих заданные программы, без изменения уже разработанного комплекса программных средств.

Эта цель достигается с помощью специального компонента — пула процессоров, — включенного в архитектуру системы. Подобно тому как создаются процесс-машины в системе SWARD, в системе iAPX 432 отдельные процессы из центральной очереди закрепляются за выделяемыми из пула процессорами. Предусмотрены средства для параллельного обслуживания нескольких процессов, для обеспечения работы нескольких процессоров и взаимной синхронизации процессов и процессоров.

Третья цель разработки данной архитектуры — создание системы высокой надежности, предоставляющей, в частности, возможность программам обрабатывать обнаруживаемые в процессе функционирования как аппаратные, так и программные ошибки. Кроме того, такая система должна допускать возможность изменения собственной конфигурации с подключением дополнительных процессоров, работающих независимо от обрабатываемых программ и выполняющих контролирующие функции.

Четвертая цель разработки системы iAPX 432 — обеспечение средств программирования на языке Ада. Структура программ для системы iAPX 432 аналогична структуре программ на языке Ада (в системе iAPX 432 имеются, например, такие понятия, как пакет и спецификатор типа объекта). Часть средств, связанная с управлением процессами, специально ориентирована на режим обработки задач, запрограммированных на языке Ада.

## АРХИТЕКТУРА СИСТЕМЫ iAPX 432

Ниже перечислены важнейшие характеристики архитектуры системы iAPX 432. Многие из них схожи с соответствующими характеристиками системы SWARD; существенное отличие состоит в том, что система iAPX 432 не располагает теговой памятью.

### ОБЪЕКТЫ

*Объект* — это основное понятие архитектуры системы iAPX 432. В гл. 4 показано, что объект представляет собой логически взаимосвязанную совокупность данных с набором применяемых к ним операций (в основном машинных команд). Как и в системе SWARD, структура памяти системы iAPX 432

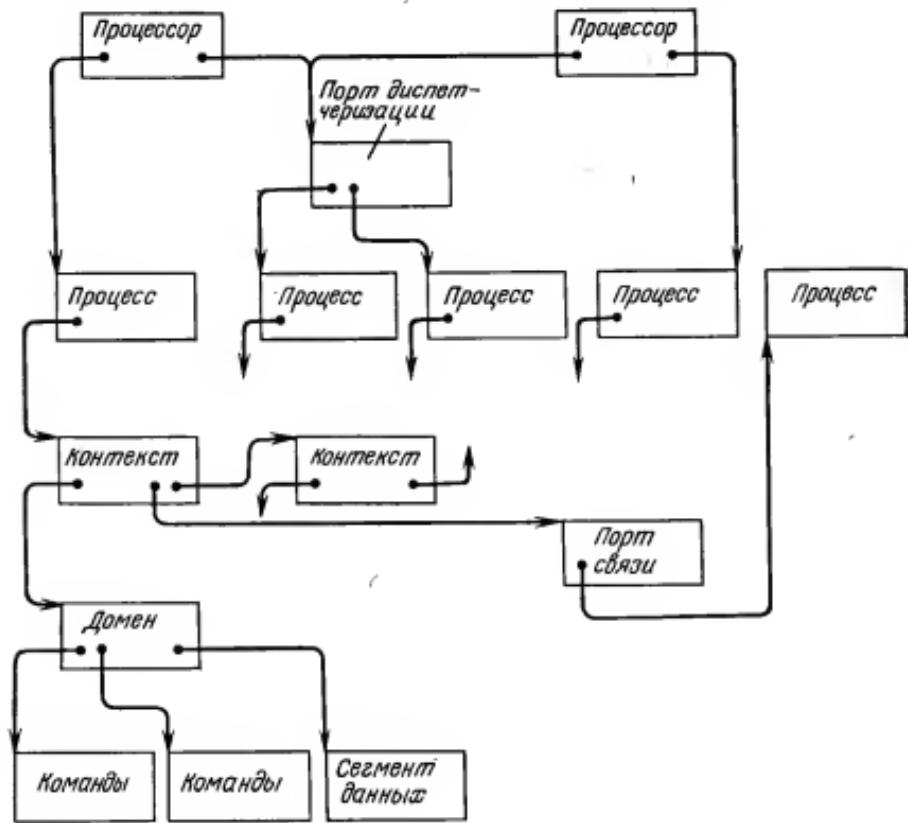


Рис. 16.1. Основные объекты системы и их взаимосвязи.

может быть представлена в виде сети объектов. Характерные объекты и возможные связи между ними изображены на рис 16.1.

Аппаратные средства системы iAPX 432 включают в свой состав объекты следующих типов: домен, порт связи, процессор, ресурсы памяти, аффинаж, транспортер, контекст, порт диспетчеризации, справочник таблиц объектов, описание типа, данные общего назначения, команды, процесс, таблицу объектов, управление дескриптором, доступ к данным общего назначения.

Перечисленные объекты являются теми объектами системы iAPX 432, которые распознаются *процессором обработки данных общего назначения* (general data processor — GDP), называемым в дальнейшем сокращенно *процессором общего назначения* и являющимся главным процессором этой вычислительной системы. Рассмотрение архитектуры указанного процессора со-

ставляет основное содержание данного раздела. В настоящее время система iAPX 432 включает также другой процессор — *интерфейсный* (interface processor — IP), служащий для сопряжения системы с «внешним миром». Интерфейсный процессор расширяет архитектуру системы iAPX 432, вводя в ее состав следующие дополнительные объекты: IP-процессор, IP-контекст, IP-процесс.

Хотя типы объектов подробно рассматриваются в гл. 17, полезно здесь кратко их описать и сравнить с соответствующими объектами системы SWARD.

Объект «команды» состоит из последовательности команд, обычно представляющей собой подпрограмму. Объект «домен» в общем случае ссылается на ряд объектов «команды» и на другие объекты, статически связанные с объектами «команды». Домен с набором объектов «команды» может быть использован для представления структуры, называемой на языке Ада пакетом; в системе SWARD эквивалентом такой структуры является объект «модуль».

Объект «контекст» создается при обращении к подпрограмме и описывает особенности активации подпрограммы и ее локальные (внутренние) переменные. Он соответствует объекту «запись активации» системы SWARD. Объект «порт связи» используется для передачи сообщений между процессами. Он схож с объектом «порт» системы SWARD, хотя в каждой системе установлены свои правила работы с этим объектом.

Объекты «порт диспетчеризации», «процессор» и «процесс» используются для управления процессами и процессорами и вместе эквивалентны объекту «процесс-машина» системы SWARD. Каждый аппаратно реализуемый процессор описывается объектом «процессор», а каждый из параллельных потоков команд (например, задачи на языке Ада) — объектом «процесс». Порт диспетчеризации осуществляет связь между объектами этих двух типов. Если конфигурация системы такова, что в ее состав входят один порт диспетчеризации, несколько процессоров и процессы, число которых превышает число процессоров, то порт диспетчеризации можно рассматривать как очередь процессов, готовых к выполнению. Из порта диспетчеризации процессы выбираются процессорами независимо (в соответствии со значениями управляющих параметров посредством нескольких программ-планировщиков) по мере того, как процессоры освобождаются для обработки очередных процессов.

Объекты «справочник таблиц объектов» и «таблица объектов» используются для перевода потенциальных адресов в физические адреса памяти. Объект «ресурсы памяти» служит для описания областей физической памяти, доступных для распределения объектам. Аналогичные средства существуют и в си-

стеме SWARD, где они реализуются аппаратно при создании конкретной модели машины и не описываются как элемент архитектуры.

Архитектурой системы iAPX 432 предусмотрено, что программы могут порождать свои собственные задаваемые пользователем объекты и снабжать их информацией о типах, с тем чтобы использование объектов было локализовано в определенных частях программы (например использование только программой обработки типов). Объект, тип которого определяется пользователем, называется *объектом расширенного типа*, а задаваемая пользователем информация о типе содержится в отдельном объекте «описание типа».

Объекты двух типов известны под общим названием «преобразователи», поскольку доступ к ним позволяет выполнять определенные операции над объектами (они фактически расширяют возможности потенциальной адресации). Объект «управление дескриптором» предоставляет возможность создавать объекты, распознаваемые аппаратными средствами (т. е. идентифицируемые как объекты этих средств), а также расширять права доступа, отражаемые в потенциальных адресах. Объект «аффинаж» позволяет формировать системные объекты, представляющие собой подмножества существующих объектов.

Объект «транспортер» используется объектами некоторых других типов для установления связи этих объектов с одной из нескольких очередей. Объекты «данные общего назначения» и «доступ к данным общего назначения» являются участками памяти, не содержащими указания типа хранимой в них информации и используемыми для хранения данных (например, значений переменных какой-либо программы пользователя).

Основное различие между объектами систем SWARD и iAPX 432 состоит в том, что объекты системы SWARD являются абстракциями, скрывающими от программиста фактическое содержание этих объектов (которое оказывается недоступным при программировании), в то время как все объекты системы iAPX 432 находятся в «поле зрения» программ (архитектура обеспечивает поразрядное представление всех объектов). Следствием этого является то, что объекты системы iAPX 432 в основном формируются действиями программы, а не машинными командами. Например, для того чтобы создать объект «процесс», программа форматизирует с помощью обычных команд обработки данных одну или несколько областей данных в соответствии со структурой объекта «процесс». Затем она сигнализирует машине (при соответствующих полномочиях программы) о том, что можно начинать рассматривать эту область как объект «процесс». Другими словами, объекты системы iAPX 432 представляют собой структуры данных с явно заданным описа-

нием, согласно которому аппаратные средства системы выполняют соответствующий набор операций. В то же время средства операционной системы (спецификаторы типов объектов) позволяют расширить ориентацию архитектуры, представляющей совокупностью объектов, трансформируя ее тем самым в архитектуру более высокого уровня.

Набор команд может быть поделен на две категории: 1) команды, оперирующие объектами определенного типа (такие, как команды SEND, CALL-CONTEXT и READ-PROCESS-CLOCK), и 2) команды, выполняющие операции над цепочкой байтов объекта любого типа (например, MOVE-INTEGER или EXTRACT-ORDINAL).

#### ПОТЕНЦИАЛЬНАЯ АДРЕСАЦИЯ

В системе iAPX 432 используется принцип потенциальной адресации и защиты от несанкционированного доступа. Потенциальный адрес (дескриптор доступа) относится к объекту и содержит два независимых набора кодов санкционированного доступа к нему. Один набор кодов доступа относится к возможности оперировать объектами как данными (например, возможность проверять и изменять структуру данных, представляющих некоторый объект). Другой набор кодов доступа касается возможности выполнять команды, производящие операции над объектом, такие, как пересылка объекта в порт.

Как указывалось в гл. 4, использование потенциальной адресации нуждается в средствах защиты от произвольного формирования потенциальных адресов с последующей манипуляцией ими. В машине с теговой памятью, такой, как SWARD, решение этой проблемы достаточно очевидно: потенциальный адрес рассматривают как теговые данные особого типа.

В системе iAPX 432 потенциальные адреса защищены путем использования списков потенциальных адресов. Объекты составляются из *сегментов*, представляющих собой смежные области физической памяти. Различают сегменты двух типов: *сегменты данных*, которые могут содержать любые данные, кроме потенциальных адресов, и *сегменты доступа*, содержащим которых могут быть только потенциальные адреса. Сегменты доступа используются только для адресации, а обращаться к ним можно лишь с помощью нескольких специальных команд, что обеспечивает защиту потенциальных адресов в системе iAPX 432.

#### «СБОР МУСОРА» В ПАМЯТИ

В системе iAPX 432 предусмотрены средства повторного запроса памяти, предназначенный для объектов. Для объектов с коротким временем существования механизм такого запроса

похож на аналогичный механизм системы SWARD. Для объектов же с продолжительным временем существования используется совершенно другой принцип учета и перераспределения памяти. В системе iAPX 432 участки памяти для объекта могут запрашиваться как из области, выделенной для процесса (объект относительно короткого времени существования), так и из области глобальных ресурсов памяти системы (объект длительного времени существования). Объекты с коротким временем существования, хотя и связаны с создаваемым контекстом (активацией подпрограммы), уничтожаются автоматически. В то же время в системе SWARD они связаны с процесс-машиной. Если в системе SWARD требуется явное уничтожение объектов продолжительного времени существования с помощью команды DESTROY, причем не исключена возможность потери объектов, то в системе iAPX 432 предусмотрены средства, позволяющие программе «сбора мусора», выполняяющей параллельно с другими программами, выявлять области памяти, относящиеся к неиспользованным объектам, и включать их в пул свободной памяти. Таким образом, в системе iAPX 432 действует принцип неявного уничтожения объектов, возможность адресации к которым утеряна. (Неадресуемые объекты перед уничтожением обычно обрабатываются так называемым спецификатором типа объекта.)

#### ОБЛАСТИ САНКЦИОНИРОВАННОГО ДОСТУПА

В момент вызова объекта «команда» создается объект «контекст». Помимо выполнения других функций, объект «контекст» определяет для объекта «команда» сферу адресации, в которую входят: 1) локальные переменные; 2) объекты, к которым обращается объект «домен», связанный с объектом «команда» (например, объекты, содержащие глобальные переменные для программы, константы и статические переменные для домена); 3) «сообщение», передаваемое из вызывающей подпрограммы. (Типичной является ситуация, когда сообщение является сегментом доступа, содержащим потенциальные адреса фактических параметров.) Таким образом, имеется возможность достаточно точно управлять механизмом внешних ссылок из подпрограмм (например, подпрограмма не может свободно обращаться к локальным переменным другой подпрограммы).

#### АВТОМАТИЧЕСКОЕ УПРАВЛЕНИЕ ПОДПРОГРАММАМИ

Система 432 содержит механизм обращения к подпрограммам, хотя и не такой развитый, как аналогичный механизм системы

**SWARD.** Для вызова подпрограммы с передачей параметров в сегмент доступа необходимо вставить потенциальные адреса формальных или фактических параметров и выполнить команду CALL-CONTEXT-WITH-MESSAGE, передавая сегмент доступа в качестве сообщения. При этом создается объект «контекст» с памятью, выделенной для локальных переменных. В отличие от системы SWARD здесь формирование списка параметров и инициализация значений локальных переменных не являются составной частью механизма обращения к подпрограммам и должны выполняться прикладными программами.

#### объекты «процесс» и «процессор»

В модели памяти iAPX 432 параллельно выполняемые процессы и аппаратно реализованные процессоры представлены в виде отдельных объектов. Тем самым достигается значительная гибкость описания функционирования системы: все манипуляции, которые могут быть выполнены над объектом, возможно произвести над процессом или процессором. Например, можно принять меры для защиты процессов или процессоров от несанкционированного доступа посредством потенциальных адресов, модифицировать их состояние путем изменения данных в соответствующем объекте, проверить состояние процесса или процессора как соответствующего объекта системы или переслать их как сообщения.

В системе SWARD понятия процесса и процессора обобщенно отражены в абстрактном понятии процесс-машина. В результате связь между физическими процессорами и процессорами (в частности, распределение процесс-машин между процессорами) не находит никакого отражения в программном обеспечении и зависит от конкретной машинной реализации системы. В системе iAPX 432 эти связи становятся явными и поэтому могут контролироваться средствами программного обеспечения. Здесь предусмотрены эффективные и в высшей степени гибкие средства контроля за распределением процессов между процессорами. В общем случае диспетчеризация низкого уровня (при операциях длительностью не более нескольких миллисекунд) выполняется аппаратно, но управление последовательностью выполнения процессов осуществляется на более высоком уровне — операционной системой, определяющей и изменяющей значения управляющих параметров в объектах «процесс». Такие параметры позволяют выполнять планирование, имеющее более общий, всеобъемлющий характер, чем то, которое достигается аппаратными средствами.

Как подчеркивается в гл. 4, в системе iAPX 432 имеется целый набор механизмов управления, позволяющих средствами

программного обеспечения реализовать большое разнообразие типов поведения вычислительной системы.

#### СРЕДСТВА ПЕРЕДАЧИ — ПРИЕМА СООБЩЕНИЙ

Порт связи является объектом, посредством которого процесс может посыпать отдельные сообщения другому процессу. Роль сообщения может выполнять объект системы (например, процесс), набор потенциальных адресов (сегмент доступа), им могут быть и данные (в частности, сегмент данных).

В отличие от системы SWARD средства передачи — приема рассматриваемой здесь системы работают асинхронно с сообщениями, поставленными в очередь к портам. Обслуживание может выполняться либо по принципу «первый поступил — первый обработан», либо в соответствии с системой приоритетов — обслуживается объект с наивысшим приоритетом. Предусмотрен также механизм отбора среди входов (*surrogate mechanism*), позволяющий организовать для процесса ожидание поступления сообщения от любого из некоторого множества портов. Это средство похоже на механизм, реализованный оператором SELECT в языке Ада. Применительно к системе iAPX 432 традиционное понятие прерывания не существует. Прерывания обрабатываются во внешней системе ввода-вывода и через интерфейсный процессор могут инициировать события в системе (например, пересылку сообщения).

#### ОБРАБОТКА ОШИБОК

Как и в системе SWARD, выявленные машиной особые ситуации принято называть *ошибками*. Обнаружение ошибки заставляет машину передать выполнение команды специальным программным средствам обработки ошибок, которые могут быть связаны с тем объектом «команды», в котором произошла ошибка. Машина формирует подробную информацию о встретившейся ошибке.

В отличие от системы SWARD в системе iAPX 432 предусмотрены отдельные средства обработки как ошибок процессоров, так и ошибок процессоров. При обработке ошибок процесса объектом анализа, помимо процесса, в котором обнаружена ошибка, обычно являются и другие компоненты системы, например операционная система. При появлении в некотором процессе такой ошибки, процесс (как объект) отсылается к специальному порту связи (называемому портом ошибок, который может быть входным портом специального процесса в операционной системе — программы обработки ошибок). Точно так же при появлении ошибки процессора (например, при аппаратных ошибках) процессор подключается к специальному порту

диспетчеризации, называемому портом диагностики, являющемуся входным для специального диагностического процесса операционной системы.

#### БОЛЬШОЙ РАЗМЕР АДРЕСНОГО ПРОСТРАНСТВА

Система iAPX 432 располагает обширным адресным пространством в отношении как размера физически реализуемой памяти, так и количества объектов, допустимых к использованию. В соответствии с возможностями средств кодирования потенциальных адресов максимальное количество адресуемых объектов может равняться  $2^{24}$ , тощее, такое количество сегментов может иметь система iAPX 432 (см. гл. 17).

Максимальный размер сегмента может достигать 65 536 байт (исключением являются сегменты «команды», размер которых ограничен 8192 байт). Итак, при обращении к объектам посредством потенциальных адресов максимальный размер адресуемого пространства равен  $2^{40}$  байт; при этом максимальный размер адресуемой физической памяти составляет  $2^{24}$  байт.

#### ГИБКОСТЬ АДРЕСАЦИИ К ОПЕРАНДАМ

Архитектуру системы iAPX 432 можно охарактеризовать как трехадресную с пересылкой данных типа «память-память»; программы не имеют прямого доступа к регистрам. Возможность использования разнообразных форм адресов операндов предоставляет широкий выбор различных типов косвенной адресации и позволяет обращаться к элементам данных векторного типа. Проще говоря, адреса операндов в команде содержат смещения относительно начала объекта и информацию, позволяющую определить местоположение объекта и задаваемую обычно в виде индекса к таблице потенциальных адресов.

Интересным расширением возможностей адресации к операндам является наличие необязательного к использованию стека операндов с возможностью обращения за любым операндом команды в стек вместо обращения за данными к объекту посредством задания их смещения относительно начала этого объекта. Один из сегментов, входящий в объект «контекст», и является стеком операндов. Присваивая битам машинной команды определенные значения, можно специфицировать эту команду, например, следующим образом: «Второй операнд команды — это содержимое вершины стека, а не ссылка на объект». Стек операндов, один из контекст (активацию), является сегментом данных в памяти, хотя в рассматриваемой версии системы 16 бит, которые должны располагаться на вершине стека, хранятся в одном из регистров процессора.

## РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Возможности выполнения операций над числами с плавающей точкой в системе iAPX 432 весьма широкие даже по сравнению с возможностями больших ЭВМ. Предусмотрено три типа представления чисел с плавающей точкой: посредством 32, 64 и 80 бит. Последний тип записи позволяет представлять числа величиной до  $1,2 \cdot 10^{4932}$ .

С помощью подпрограммы можно управлять способом округления (до ближайшего значения, до ближайшего большего, до ближайшего меньшего или до нуля в младшем разряде), точностью выполнения операций с плавающей точкой и обработкой неточных результатов (как при наличии ошибок, так и при их отсутствии).

## КОМПОНЕНТЫ СИСТЕМЫ И ЕЕ ВОЗМОЖНАЯ КОНФИГУРАЦИЯ

В рассматриваемую версию системы iAPX 432 входят компоненты трех типов. Два из них — 43201 и 43202 — составляют процессор общего назначения. Модуль 43201 выполняет выборку и декодирование команд машины, а большая часть операций по адресации и арифметических операций реализуется модулем 43202. Оба модуля тесно связаны друг с другом 16-разряднойшиной с микропрограммным управлением. Оба устройства построены на программируемых логических матрицах; используется большое разнообразие микропрограмм.

Модуль 43203 — это интерфейсный процессор, выполняющий роль канала ввода-вывода. Он является средством расширения набора объектов и команд системы iAPX 432 и дает возможность программам устанавливать связи с другими подсистемами через внешний интерфейс. Интерфейсный процессор позволяет процессору ввода-вывода выполнять подмножество команд iAPX 432 по передаче данных в память и из памяти системы через четыре назначаемых окна памяти<sup>1)</sup>. С его помощью общие принципы защиты данных в системе iAPX 432 распространяются и на уровень внешнего интерфейса. Так, внешние устройства не могут записывать данные в произвольные области памяти, а только в специальные объекты, создаваемые соответствующими процессами системы.

<sup>1)</sup> Речь идет о размещенных в интерфейсном процессоре программируемых ассоциативных блоках памяти, позволяющих программам путем осуществлять отображение области адресов подсистемы ввода-вывода в адреса системы iAPX 432. — Прим. ред.

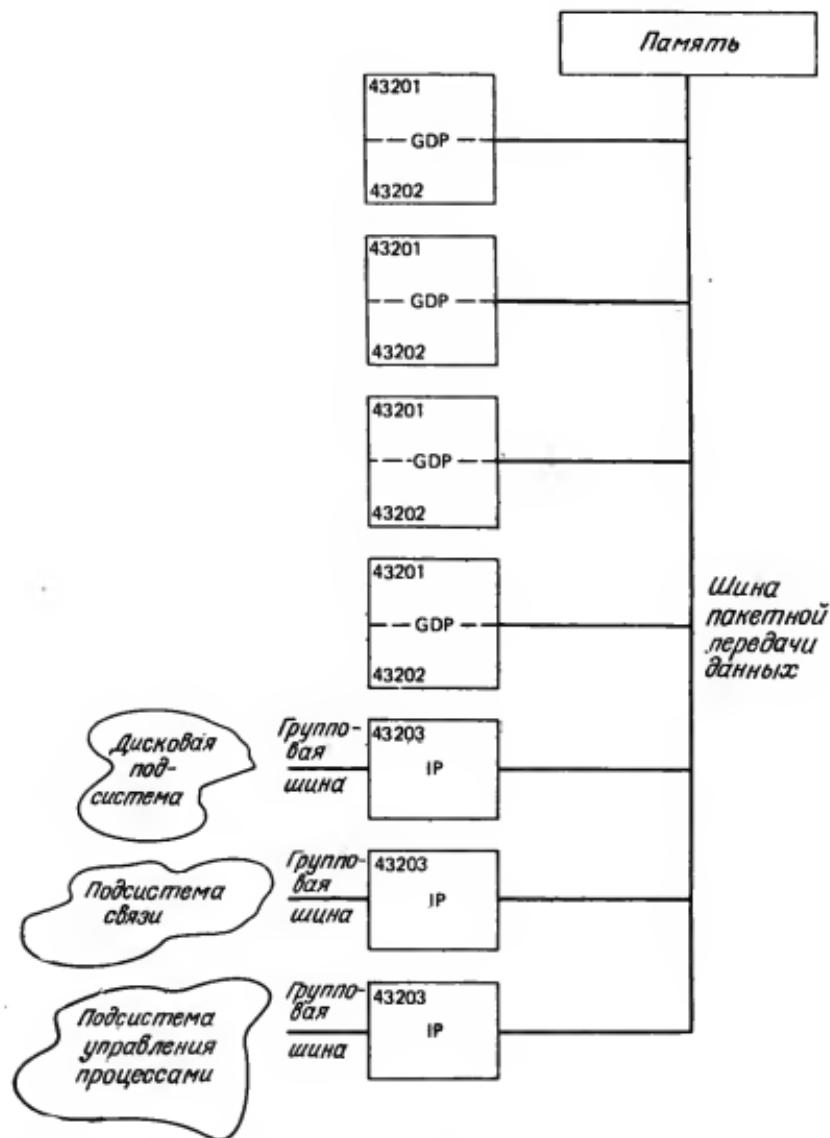


Рис. 16.2. Пример конфигурации системы iAPX 432.

На рис. 16.2 изображена конфигурация большой системы iAPX 432 с четырьмя процессорами общего назначения (GDP) и тремя интерфейсными процессорами (IP). Сопряжение с памятью осуществляется через специальную шину, ориентированную на передачу пакетов информации. При выполнении каж-

дой операции над данными из памяти может быть передано от 1 до 6 байт.

При работе нескольких процессоров особое внимание следует уделять согласованию обращения к памяти во избежание их наложений. При наличии только одной шины доступа к памяти целесообразно использование не более пяти процессоров GDP. Дальнейшее увеличение числа этих процессоров вследствие конкуренции за доступ к общей памяти вносит несущественный вклад в эффективность работы системы. Система с единственной шиной доступа и пятью процессорами GDP имеет приблизительно такую же скорость выполнения команд, как и система с тремя автономными процессорами GDP. При наличии большого числа процессоров требуется использование нескольких шин доступа, коммутируемых, например, матричными переключателями.

## ЛИТЕРАТУРА

1. iAPX 432 General Data Processor Architecture Reference Manual, Intel. Corp., Santa Clara, CA, 1981.
2. iAPX 432 Component User's Guide, Intel Corp., Santa Clara, CA, 1981.
3. iAPX 432 Interface Processor Architecture Reference Manual, Intel Corp., Santa Clara, CA, 1981.
4. Rattner J., Lattin W. W., Ada Determines Architecture of 32-bit Microprocessor, *Electronics*, 4(54), 119—126 (1981).
5. Rattner J., Architecture of the Intel iAPX 432 Micromainframe: A Personal History, *Lambda*, 2(1), 27—29 (1981).

---

## ГЛАВА 17

### ПРОЦЕССОР ОБЩЕГО НАЗНАЧЕНИЯ iAPX 432

В этой главе описывается архитектура процессора общего назначения iAPX 432, причем рассмотрение ведется в основном с точки зрения организации памяти системы. Материал излагается последовательно, от определения понятий нижнего уровня иерархии системы — сегментов и типов данных — до описания элементов самого высокого уровня — объектов. Далее обсуждается наиболее сложная по структуре и трудная для восприятия характеристика рассматриваемой архитектуры — адресация. В заключение данной главы описываются типы объектов, техника обработки ошибок и другие средства системы. Форматы команд и соответствующие операции рассматриваются в гл. 18.

#### СЕГМЕНТЫ И ТИПЫ ДАННЫХ

Для средств управления памятью и адресации системы iAPX 432 основной единицей манипулирования является не объект, а сегмент. Объекты обычно состоят из нескольких сегментов. Сегмент — это непрерывный участок физически реализованной памяти размером от 1 до 65 536 байт. Перечислим пять основных понятий, характеризующих сегменты:

- 1) *базовый адрес*, являющийся 24-битовым адресом первого байта сегмента;
- 2) *длина* (1—65 536 байт);
- 3) *базовый тип* (сегментами базового типа являются сегменты данных и сегменты доступа);
- 4) *системный тип* (сегмент такого типа используется для определения объекта, например объекта «коинтекст» или «процесс», частью которого является этот сегмент);
- 5) *дескриптор памяти в таблице объектов*, содержащий атрибуты сегмента (физический адрес, длину, тип) и используемый для преобразования потенциальных адресов.

Различие между сегментами доступа и данных заключается в том, что первые могут содержать только потенциальные адреса (дескрипторы доступа), а вторые — все, кроме потенциальных адресов. Машине не допускает использования сегмента доступа в качестве данных (регистрируется ошибка) или какой-либо части сегмента данных в качестве потенциального адреса.

Следствием этого является то обстоятельство, что, например, для записи на языке Ада фрагмента

```
type LIST_ELEM is
  record
    CHAIN : access LIST_ELEM;
    VALUE : INTEGER;
    NAME : STRING(1..16);
  end record;
```

возникает необходимость в использовании нескольких сегментов: сегмента данных для размещения значений переменных VALUE и NAME и представляющем эту запись сегмента доступа для размещения потенциального адреса CHAIN и потенциального адреса упомянутого сегмента данных.

При переходе от понятия «сегмент» к другим понятиям, используемым при описании системы iAPX 432, необходимо помнить следующее:

1. Все, принадлежащее модели памяти системы iAPX 432, описывается в терминах «сегменты»: переменные в программе — это строки байтов, смещенные на некоторую определенную величину относительно начала сегмента и в нем расположенные; объекты — это структуры данных одного или нескольких сегментов (приведенная выше запись LIST\_ELEM — двухсегментный программно-определяемый объект).

2. Потенциальные адреса служат по сути дела для обращения к сегментам, а не к объектам. Однако, как и в рассмотренном выше примере, многосегментные объекты включают в свой состав *корневой сегмент доступа*, содержащий ссылки на другие сегменты. Часто потенциальный адрес такого корневого сегмента называют *потенциальным адресом объекта*.

Поскольку система отличает сегменты доступа от сегментов данных, а также в связи с тем, что сегменты доступа могут содержать только потенциальные адреса, можно утверждать, что iAPX 432 является в определенном смысле машиной с теговой памятью, располагающей теговыми данными двух типов: строками байтов и векторами потенциальных адресов. При этом, как и можно было ожидать, рассматриваемая система

допускает различную интерпретацию строки байтов. Прежде чем перейти к этому вопросу, отметим, что упомянутая интерпретация, т. е. описание типов строк байтов, совпадает по смыслу с описанием типов данных архитектуры машины с теговой организацией памяти, например такой, как SWARD. Рассмотрение типов данных системы iAPX 432 по сути дела сводится к рассмотрению набора команд. Список и формат типов данных приведены на рис. 17.1. Так называемые *символьные данные* занимают 1 байт памяти и используются для представления целого числа без знака, символа (соответствующего одному из стандартных символов, представляемых в коде

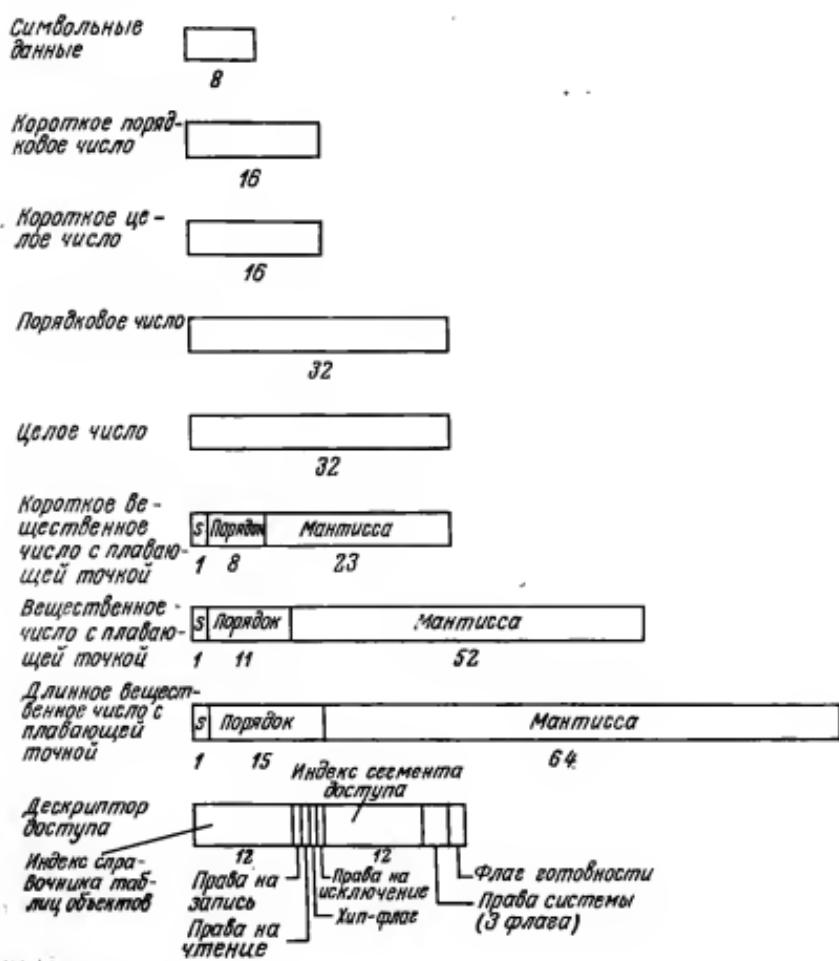


Рис. 17.1. Типы данных системы iAPX 432.

ASCII) или логической величины (для представления которой используется только бит младшего разряда). Имеются два типа данных, именуемых *порядковые числа* и являющихся двоичными целыми числами без знака. Они часто используются в качестве значений индексов, содержимого счетчиков или строк битов. Имеются также два типа данных *целого типа*, представляемых в традиционной форме в виде дополнительных кодов. Числовые данные в форме с плавающей точкой представлены тремя типами, которые отличаются не только точностью описания мантиссы, но и величиной порядка (задаваемого со смещением 127, 1023 или 16 383). Знак такого числа задается содержимым двоичного знакового разряда. Первые два типа вещественных чисел (длиной 32 и 64 бит) используют такую форму нормализованного представления мантиссы, при которой ее ведущий значащий бит находится за разрядной сеткой. Благодаря этому вещественные числа короткой или нормальной длины представляются с точностью на один двоичный разряд более высокой, чем точность при традиционных формах представления чисел с плавающей точкой. Если поле порядка вещественных чисел любого из трех рассматриваемых типов содержит единицы во всех двоичных разрядах, это указывает на отсутствие числа (так называемое представление NaN — not a number).

Все арифметические операции с плавающей точкой процессор выполняет над числами, представленными в виде так называемых временных данных вещественного типа. Как упоминалось в гл. 16, объект «контекст», связанный с активацией подпрограммы, содержит спецификацию требуемых характеристик операций с плавающей точкой (параметры округления, точность представления результатов, действия при отсутствии требуемого результата).

Система iAPX 432 имеет в своем распоряжении данные еще одного типа, называемые *дескриптором доступа*, или *потенциальным адресом*. Дескриптор доступа используется как средство обращения к сегменту и определения прав доступа к последнему. Формат дескриптора доступа предполагает наличие следующих полей: *индекса справочника таблиц объектов*, *индекса сегмента* (используемого для определения адреса дескриптора объекта, адресуемого данным дескриптором и описываемого более подробно в последующих разделах), *флагов прав объекта*, *флагов прав системы*, *хип-флага (heap flag)* и *флага готовности*. Последний указывает, можно ли использовать данный дескриптор доступа для адресации.

Различие между «правами объекта» и «системными правами» заключается в том, что флаги прав объекта указывают, какие операции можно выполнить над сегментом, рассматривая

его просто как сегмент данных или доступа, в то время как флаги прав системы указывают, какие операции можно выполнить над сегментом как над системным объектом (или корневым сегментом доступа). «Права объектов» представлены в дескрипторе доступа следующими флагами: чтения, записи и исключения, или удаления (возможность удалить сам дескриптор доступа или записать на его месте другую информацию). Конкретная интерпретация флагов прав системы зависит от типа адресуемого объекта. Назначение хип-флага, связанного с работой системы управления памятью, описывается в следующем разделе.

## СЕГМЕНТЫ И ОБЪЕКТЫ

Система iAPX 432 базируется на принципе наложения на сегменты масок, имеющих специальные, заранее определенные форматы, с целью создания объектов и задания набора машинных операций над этими объектами. Объектом может быть одиночный сегмент, набор взаимосвязанных сегментов (обычно использующих корневой сегмент доступа для связи с другими сегментами) или часть сегмента, называемая *аффинажем* (*refinement*).

Главное отличие архитектуры системы iAPX 432 от архитектуры системы SWARD состоит в том, что пользователю первой системы доступна структура объектов, в то время как для пользователя второй системы объекты — это некоторые абстрактные понятия, а структура этих объектов имеет конкретное воплощение только при реализации машины. Архитектуры этих систем различаются также тем, что все объекты системы iAPX 432 (за исключением объекта «контекст») конструируются в соответствии с алгоритмами программ, а в системе SWARD объекты создаются путем выполнения машинных операций. В системе SWARD анализ состояния объекта осуществляется путем выполнения машинной команды, например команды DESCRIBE-CAPABILITY. В системе iAPX 432 подобная цель достигается путем анализа данных внутри объекта, что не обязательно предполагает «осведомленность» программ системы iAPX 432 о структуре объектов; к обсуждению этого вопроса мы еще вернемся.

Для иллюстрации сказанного выше обратимся к объекту «порт», схематически изображенному на рис. 17.2 (порты диспетчеризации и связи имеют одинаковую структуру). Порт состоит из сегмента доступа и сегмента данных. (Потенциальный адрес, которым является дескриптор доступа, называют потенциальным адресом объекта «порт»). Порт может создаваться следующим образом. Сначала формируются два сегмента. По-

скольку они являются частью системного объекта, т. е. объекта, определенного системой, а не пользователем, для их создания используются не обычные команды формирования сегментов (CREATE-ACCESS-SEGMENT и CREATE-DATA-SEGMENT), а команда CREATE-TYPED-SEGMENT. (Как следует из рассматриваемой в дальнейшем проблемы защиты потенциальных адресов, эта команда не может быть использована односторонне, т. е. без взаимного обращения к ней адресата.) Затем полям обоих сегментов присваиваются начальные значения:

*байт*



*байт*



Рис. 17.2. Представление объекта «порт».

ния в соответствии с заданным исходным состоянием. Это осуществляется посредством обычных команд манипуляции данными, причем в сегмент доступа будет помещен дескриптор доступа (ДД) для обеспечения обращения к сегменту данных.

Оставим на некоторое время обсуждение специфики объектов типа «порт» и сделаем несколько общих замечаний относительно сегментов на основании анализа рис. 17.2. Прежде всего отметим, что у сегментов нет заранее установленных, фиксированных размеров. Это же справедливо для некоторых, но не для всех объектов. Порт представляет собой очередь: порт связи — очередь сообщений или принятых процессов, порт диспетчеризации — очередь процессов или не занятых выполнением операций процессоров. Верхняя часть адресного пространства обоих сегментов (т. е. часть с наибольшими значениями адресов) используется для хранения информации об очередях. Благодаря этому программа, создающая порт, имеет возможность управлять размером очереди. В некоторых объектах (но не в рассматриваемом случае) определенное функциональное назначение предписывается только нижней части сегментов (т. е. областям памяти с наименьшими номерами), так что по-

является возможность использования оставшейся части сегмента (сегментов) программ для размещения их собственной информации, если имеется такая необходимость.

Следует отметить также наличие в сегменте данных (рис. 17.2) поля, называемого *блокировка объекта* (*object lock*). Это поле используется как программным обеспечением системы (для синхронизации использования объекта одновременно протекающими процессами), так и аппаратными средствами (для предотвращения попыток одновременного обновления искаженными процессорами информации в одном и том же объекте).

Напомним, что в дескрипторе доступа имеется поле «права системы». Интерпретация содержимого этого поля зависит от типа адресуемого объекта. Так, для порта связи первый бит указывает, разрешается ли посыпать в этот порт сообщение; второй бит сигнализирует, можно ли принимать сообщение из даниого порта; третий бит не используется.

#### СПЕЦИФИКАТОРЫ ТИПОВ ОБЪЕКТОВ

Хотя формат объекта «порт» определяется при описании архитектуры системы и маловероятно внесение изменений в его определение, представляется естественным лишить программы информации о формате такого объекта. Предпочтительнее такое решение, когда программы манипулируют портом как абстрактым понятием, для выполнения операций над которым задан набор некоторых функций. Некоторые из этих функций реализуются отдельными машинными командами (в частности, командами SEND, RECEIVE), другие — программными модулями (например, подпрограммой CREATE\_PORT).

Тогда естественно предположить наличие в системе для объекта каждого типа (включая объекты, тип которых определяется программистом) так называемого *программного спецификатора типа объекта* (*type manager*). Последний представляет собой набор взаимосвязанных подпрограмм, реализующих внутреннюю структуру объекта определенного типа путем обеспечения выполнения над объектом операций в соответствии с его типом. При этом для программ, выполняемых системой, объект представляется неким «черным ящиком» — абстрактым понятием, конкретная структура которого неизвестна. Спецификаторы системных объектов обычно являются частью операционной системы. Так, подпрограмма CREATE\_COMMUNICATION\_PORT была бы подпрограммой операционной системы. Делая «невидимой» для программ пользователя информацию о формате объекта «порт», подобная подпрограмма в результате обращения к ней создает дескриптор доступа с полным набором разрешенных форм доступа (прав) системы, но не объекта.

Функции спецификаторов типов объектов описываются в нескольких последующих разделах, поскольку в системе iAPX 432 имеются мощные программные средства для реализации этих функций.

### АФФИНАЖ

Аффинаж — это объект, представляющий собой последовательность смежных байтов в пределах некоторого сегмента. Несколько иначе такой объект можно рассматривать как дескриптор доступа к некоторой части сегмента, но не ко всему сегменту. Положим, имеется сегмент данных, содержащий локальные переменные активной подпрограммы. Пусть при этом требуется создать дескриптор доступа к какой-либо одной переменной внутри сегмента. Возможна ситуация, когда эта локальная переменная — один из нескольких фактических параметров, подлежащих передаче в подпрограмму посредством адреса. Желательно поместить дескрипторы доступа к конкретным параметрам в сегмент доступа и передать последний, называемый сообщением, вызываемой подпрограмме. При таком решении дескрипторы доступа будут адресоваться к аффинажам сегмента данных и могут быть созданы командой CREATE-GENERIC-REFINEMENT, которая по заданным дескриптору доступа к сегменту, смещению и длине создает соответствующий дескриптор доступа к подпространству (части) сегмента. Следовательно, эта команда подобна команде COMPUTE-CAPABILITY системы SWARD.

Отметим, что аффинажи необходимы для организации доменов — небольших областей адресного пространства, защищенных от несанкционированного доступа (см. гл. 4). Следует помнить, что дескриптор доступа к аффинажу нельзя использовать для обращения к другим данным того же сегмента, поскольку механизм адресации обращается с аффинажем как с отдельным сегментом (т. е. у него имеются базовый адрес и длина). В действительности же при наличии дескриптора доступа к аффинажу существует способ получения дескриптора доступа ко всему сегменту. Для этой цели используется команда RETRIEVE-REFINEDOBJECT, которая, однако, может быть защищена от несанкционированного доступа.

### АДРЕСАЦИЯ

Система адресации, т. е. средства обращения команд к своим operandам, — это наиболее сложное и трудное для понимания понятие архитектуры системы iAPX 432. Сложность системы адресации обычно представляет интерес только для разработ-

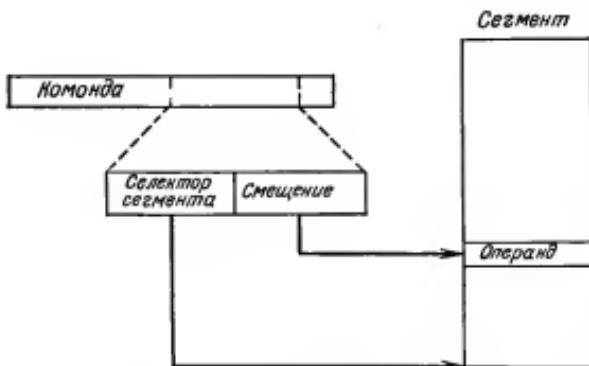


Рис. 17.3. Уровень наибольшей абстракции адресации к данным.

чиков компиляторов. Поэтому знакомство с адресацией следует осуществлять постепенно, переходя от одного уровня ее структуры к другому. Начав с уровня наибольшей абстракции, мы перейдем потом к уровню, описывающему детали структуры адресации.

С позиций уровня наибольшей абстракции обращение машинных команд к одному или нескольким операндам внутри сегмента соответствует схематическому представлению, приведенному на рис. 17.3. Поле обращения к данным (адреса операнда) команды содержит информацию, используемую для определения местоположения сегмента, а также информацию о смещении, предназначенную для определения положения конкретных данных на границе байтов внутри сегмента. Команды заранее не «привязаны» к конкретным сегментам и физическим

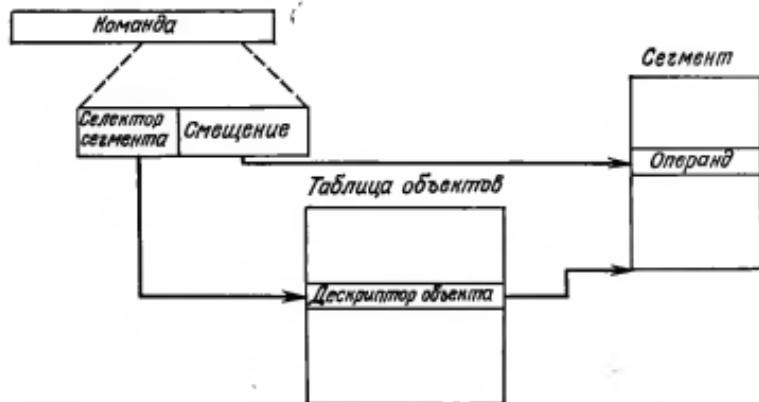


Рис. 17.4. Адресация посредством таблицы объектов.

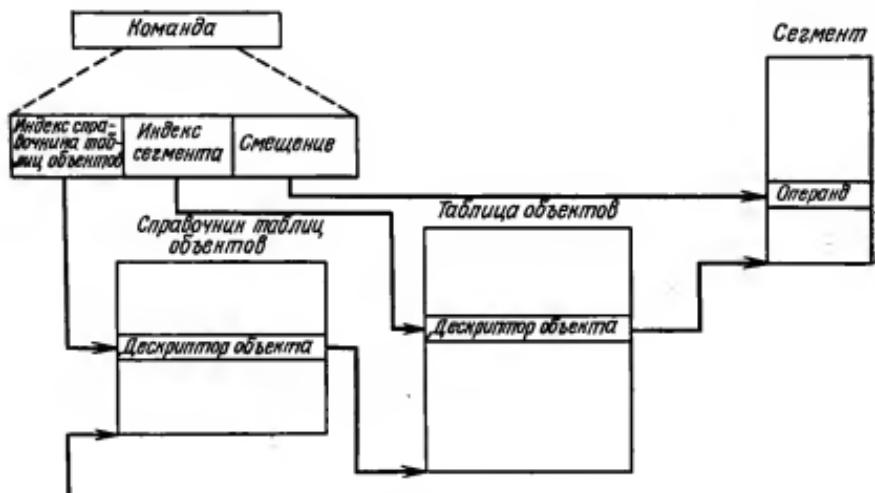


Рис. 17.5. Структура двухуровневой таблицы объектов. (Обычно в системе предусмотрен только один справочник таблиц объектов; адрес базы известен всем процессорам посредством объекта «процессор».)

адресам сегментов. Более того, имеется много уровней косвенных взаимосвязей между командой и ее операндом.

Другим, менее высоким уровнем абстракции структуры адресации является адресация посредством таблицы объектов (рис. 17.4). Каждый сегмент располагает ассоциативной памятью, представленной дескриптором объекта в таблице объектов. Этот дескриптор определяет физический адрес сегмента, его длину, базовый тип (сегмент данных или сегмент доступа) и тип его объекта. Поскольку максимальное число сегментов равно  $2^{24}$ , таблица объектов имеет двухуровневую структуру (рис. 17.5). Объект «справочник таблиц объектов» — это таблица объектов таблиц объектов. Каждая таблица объектов описывается дескриптором объекта в справочнике таблиц объектов.

Для адресации сегмента используются два индекса. Первый индекс — 12-битовый индекс справочника — указывает на элемент в справочнике таблиц объектов, который является ссылкой на таблицу объектов. Второй индекс — 12-битовый индекс сегмента — указывает на дескриптор объекта в этой таблице объектов, который содержит ссылку на искомый сегмент. Согласно представленному на рис. 17.1 описанию дескриптора доступа (потенциального адреса), в нем имеются индексы справочника и сегмента. Следовательно, ссылка на данные в команде является обращением к дескриптору доступа и содержит информацию о смещении.

Обычно в системе имеется только один справочник таблиц объектов. Количество этих таблиц определяется программным обеспечением: обычно операционная система создает одну таблицу объектов для каждого активного процессса.

Рис. 17.6 иллюстрирует еще несколько уровней структуры адресации команды к своим операндам. Поле обращения к данным должно содержать ссылку на дескриптор доступа и величину смещения (адресуемых данных относительно начала сегмента). Выбор дескриптора доступа осуществляется посредством ссылки, состоящей из двух частей: *селектора* и *индекса входного сегмента доступа* (ВСД).

Объект «контекст» — один на каждую активацию подпрограммы — определяет то адресное пространство, которое «окружает» подпрограмму, т. е. пространство, с которым она взаимодействует. В любой момент времени может быть не более четырех сегментов доступа, которые содержат ДД, непосредственно используемые командами. Эти сегменты называются

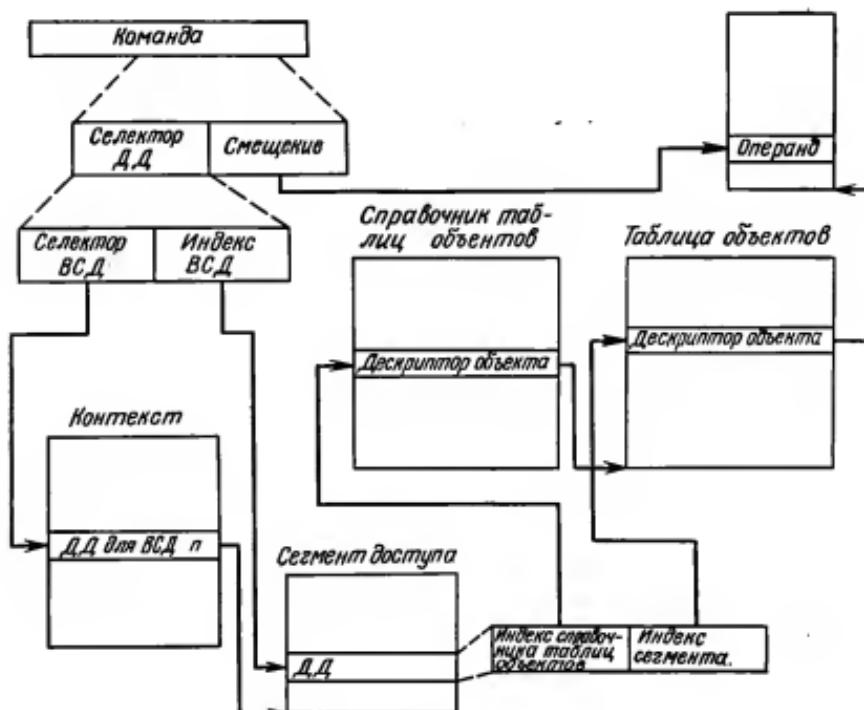


Рис. 17.6. Путь от обращения к операнду в команде до местоположения операнда.

входными сегментами доступа, поскольку именно они в данный момент используются при адресации.

Одним из входных сегментов доступа, обозначаемым ВСД 0, является сегмент, выбираемый селектором ВСД со значением 0 и являющийся корневым сегментом доступа самого контекста. Другими тремя из четырех ранее упомянутых сегментов доступа являются любые сегменты, адресуемые самим объектом «контекст». Для программного обеспечения возможно принятие следующего соглашения: селектор 0 обращается к сегменту доступа к контексту, селектор 1 — к сегменту доступа, обращающемуся к общим, или глобальным, данным, селектор 2 — к сегменту доступа к текущему сообщению (например, списку параметров), селектор 3 предназначен для общего использования.

На данной стадии анализа адресации можно утверждать, что имеется по крайней мере семь уровней косвенных взаимосвязей между командой и ее операндом, поскольку по крайней мере дважды необходимо пользоваться справочником таблиц объектов: один раз для извлечения данных из контекста с передачей в выбраный входной сегмент доступа и другой раз для извлечения из выбранного дескриптора доступа с передачей в сегмент данных. Перечислим действия, выполняемые на указанных уровнях:

- 1) селектор ВСД в команде выбирает ДД для входного сегмента доступа;
- 2) индекс справочника из этого ДД выбирает дескриптор объекта из справочника таблиц объектов;
- 3) дескриптор объекта выбирает таблицу объектов, которая совместно с индексом сегмента из ДД позволяет выбрать сегмент; последний подлежит использованию в качестве входного сегмента доступа;
- 4) индекс ВСД из команды выбирает ДД из этого входного сегмента доступа;
- 5) индекс справочника из этого ДД выбирает дескриптор объекта из справочника таблиц объектов;
- 6) дескриптор объекта выбирает таблицу объектов, которая совместно с индексом сегмента из ДД позволяет выбрать искомый сегмент;
- 7) смещение из команды обеспечивает выбор искомых данных из этого сегмента.

Далее будет указано, что селектор ВСД и индекс не всегда размещаются непосредственно в команде, а это приводит к образованию еще одного или двух дополнительных уровней взаимосвязи команды со своим операндом.

Наличие указанных косвенных взаимосвязей обычно не влечет за собой дополнительные временные издержки при адресации. Это объясняется следующими двумя обстоятельствами.



Рис. 17.7. Два типа обращения к операндам.

Во-первых, локальные переменные как наиболее часто адресуемые данные обычно расположены в сегменте данных, связанном с объектом «контекст». Одного, а не двух проходов по таблице объектов обычно достаточно для определения местоположения этих данных. Во-вторых, каждый процессор имеет несколько ассоциативных запоминающих устройств, размещенных на том же кристалле, на котором находится сам процессор. А как будет описано в гл. 18, процессор хранит в этой памяти информацию, полученную от дескрипторов доступа и объекта, которые использовались последними. Поэтому процессор часто может обходиться без выполнения стандартных этапов адресации.

#### АДРЕСАЦИЯ ОПЕРАНДОВ КОМАНД

На рис. 17.6 показана последовательность адресации системы iAPX 432, начиная с получения из команды значения селектора ВСД, индекса ВСД и смещения. Однако это только вторая часть структуры адресации; ее первая часть связана с получением исходной информации для описанной второй части. Принципы формирования адреса операнда в команде схематически изображены на рис. 17.7. Поле адреса операнда в команде может содержать обращение либо к данным в сегменте, либо к данным на вершине стека операндов. Каждый объект «контекст» располагает связанным с ним сегментом данных, называемым стеком операндов. Этот стек размещается в памяти, и его размер должен быть задан компилятором. В формате команд имеется поле, содержимое которого указывает, находится ли операнд в сегменте данных или в стеке. В последнем случае в формате команды поле адреса операнда отсутствует. Многие команды имеют различные формы представления в зависимости от местоположения операндов-исходных данных и операндов-результатов, а именно: команды типа память — память, память — стек, стек — стек и т. п. Если не один, а несколько операндов команды обращаются к стеку, то обращение к вершине стека выполняется в соответствии с определенным порядком.

Рассмотрим еще один возможный формат команды. Пусть содержимое поля адреса операнда должно в результате при-

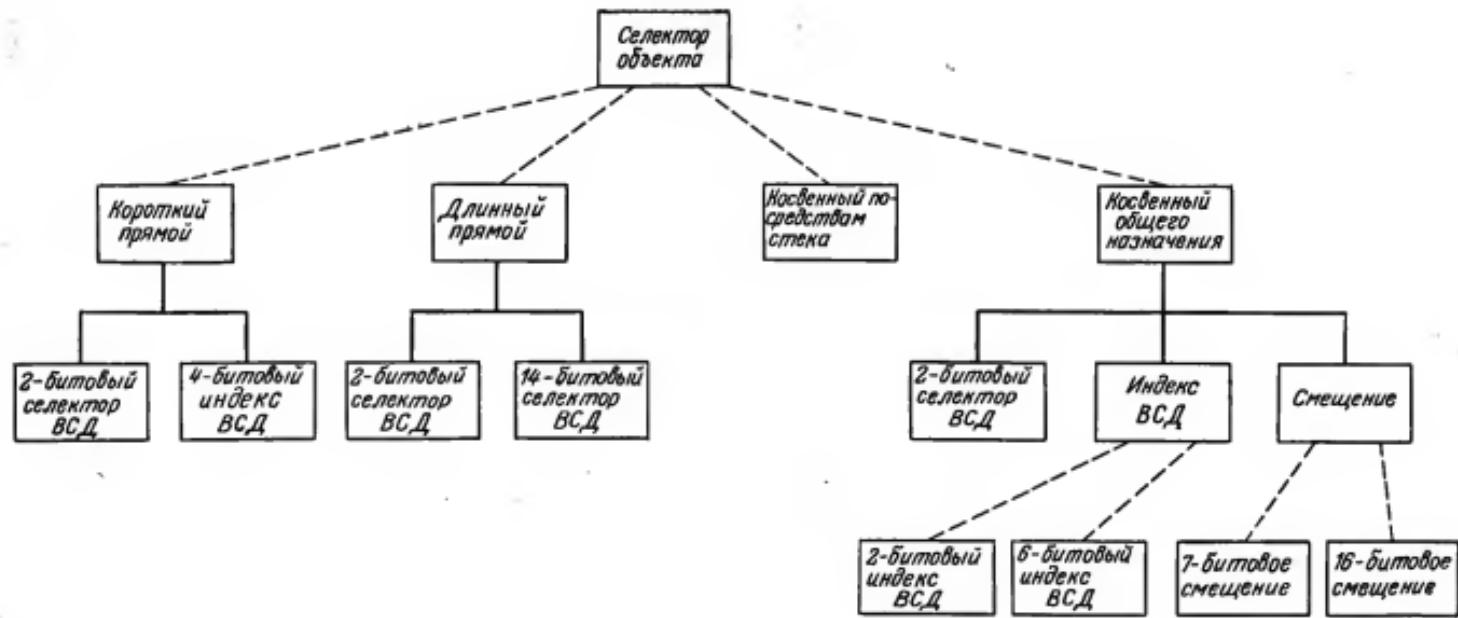


Рис. 17.8. Варианты представления селектора объекта посредством адреса операнда.

вестн к получению селектора объекта (т. е. селектора ВСД и индекса ВСД), который в соответствии со схемой, представленной на рис. 17.6, должен обеспечить обращение к сегменту и получение значения смещения. Возможны различные способы решения этой задачи.

Рис. 17.8 иллюстрирует семь возможных вариантов представления селектора объекта посредством адреса операнда в команде. Простейший вариант предполагает использование 6-битового короткого прямого селектора: 2 бит для выбора ВСД, 4 бит для индексирования одного из первых 16 дескрипторов доступа в ВСД. При втором варианте используется 16-битовый длинный прямой селектор, позволяющий индексировать один из 16 536 дескрипторов доступа в выбранном ВСД. Третий вариант связан с использованием стека. (Применение стека в этом случае не имеет ничего общего с обращением к стеку на рис. 17.7; здесь стек может использоваться как для данных, так и для адресов.) Верхние 16 бит стека интерпретируются как длинный прямой селектор.

Остальные четыре варианта предполагают получение селектора объекта косвенным образом и отличаются лишь размером некоторых полей. Поле косвенного селектора общего назначения содержит селектор ВСД, индекс ВСД и смещение (последнее не следует путать со смещением операнда). Эта информация служит для адресации (аналогично тому, как показано на рис. 17.6), но не операнда, а 16-битового длинного прямого селектора, расположенного в указанном сегменте с определенным смещением. Затем этот длинный прямой селектор используется описанным выше способом (согласно рис. 17.6) для обращения к сегменту, содержащему операнд.

В соответствии с рис. 17.7, адрес данных состоит из двух частей: селектора объекта и смещения операнда. Возможные варианты представления смещения показаны на рис. 17.9. Общее количество этих вариантов равно 79. В сочетании с семью вариантами представления селектора объекта это обеспечивает 553 различные формы представления адресов операндов в системе iAPX 432 (причем это не зависит от действительных значений адресов). К полученному количеству форм представления адресов операндов следует еще добавить обращение к стеку (см. рис. 17.7).

Возможные варианты представления смещения операнда можно разделить на четыре группы: скаляры, элементы записи, элементы вектора, элементы вектора динамического типа. Согласно архитектуре системы iAPX 432, явное представление записей в векторах отсутствует.

Первой и простейшей формой представления смещения является его представление в виде скаляра: задается 7- или 16-би-

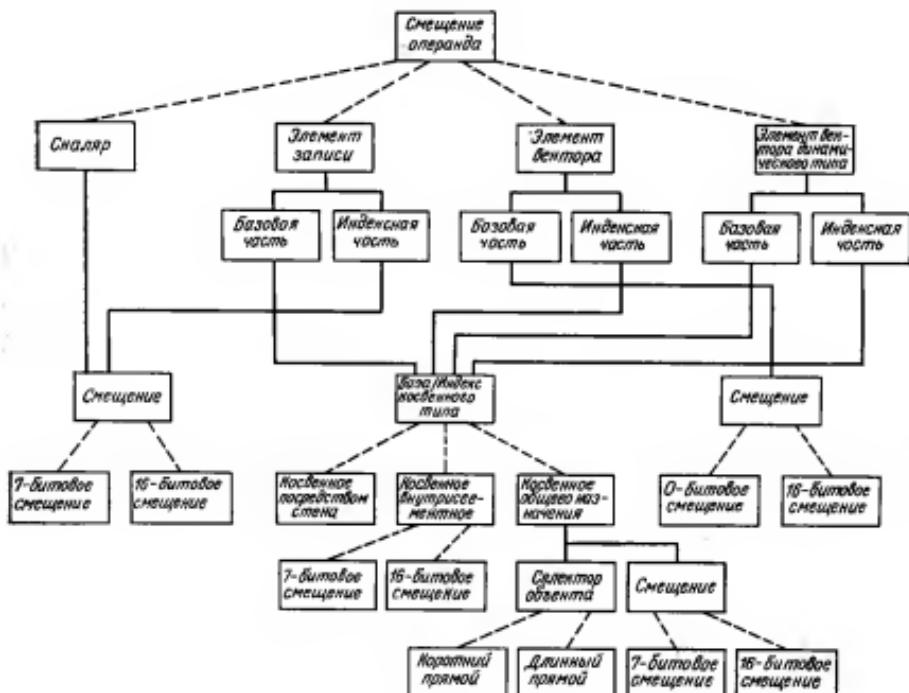


Рис. 17.9. Типы представления смещения операнда посредством адреса операнда.

товое смещение (в байтах) в выбранном сегменте. Вторая форма — это представление смещения в виде элемента записи: создаются два смещения, которые складываются перед использованием. Первое смещение — базовая часть — команда получает косвенным путем, второе смещение — индексная часть — непосредственно связано с командой в виде 7- или 16-битовой величины. Следовательно, смещение в виде элемента записи обеспечивает адресацию к содержимому сегмента, расположенному в этом сегменте с некоторым переменным смещением (определенным базовой частью) на фиксированном расстоянии от последнего (определенном индексной частью).

Возможны семь различных способов задания переменного смещения — базовой части смещения. Оно может извлекаться в виде 16-битовой величины из вершины стека (косвенная стековая адресация). Оно может считываться как 7- или 16-битовое смещение, хранимое в том же сегменте в виде операнда, к которому для этой цели необходимо адресоваться (косвенная внутрисегментная адресация). Наконец, 16-битовое базовое смещение может быть выбрано из заданного смещения в ука-

занном сегменте путем использования косвенной адресации общего назначения (рис. 17.9).

Третьей формой представления смещения является его представление в виде элемента вектора. Результирующее значение такого смещения определяется суммированием двух компонентов — фиксированной базовой части смещения (которая может адресоваться к началу сегмента или смещению внутри сегмента) и переменной, или косвенной, индексией части. Последняя может быть задана одним из семи способов косвенной адресации, подобно тому как это делается для задания базовой части смещения, представляемого в виде элемента записи.

Архитектурой системы iAPX 432 предусмотрено автоматическое масштабирование индексной части смещения перед ее суммированием с базовой частью, т. е. машина выполняет следующие вычисления:

$$\begin{aligned} \text{Смещение} = & \text{Базовая часть} + (\text{Масштабный коэффициент} \times \\ & \times \text{Индексная часть}). \end{aligned}$$

Масштабный коэффициент может принимать значения 1, 2, 4, 8 и 16. Выбор конкретного значения определяется неявно в соответствии с типом команды. Например, при использовании команды ADD-REAL масштабный коэффициент равен 8, поскольку эта команда оперирует вещественными числами с плавающей точкой (8-байтовыми операндами).

Четвертая форма представления смещения получила название «смещение в виде элемента вектора динамического типа». Она подобна третьей форме представления смещения и отличается от нее лишь тем, что значения обеих частей смещения (базовой и индексной) определяются посредством косвенной адресации.

#### АДРЕСАЦИЯ ДЕСКРИПТОРОВ ДОСТУПА И ОБЪЕКТОВ

Декодирование содержимого поля операнда команды позволяет установить, является ли operand дескриптором доступа или объектом. После этого для непосредственного доступа к операнду требуется пройти еще несколько этапов (уровней) косвенной адресации. Более подробно этот вопрос рассматривается в гл. 18.

#### АДРЕСАЦИЯ С ЦЕЛЬЮ ПЕРЕДАЧИ УПРАВЛЕНИЯ

В командах передачи управления имеется адрес особого типа, называемый адресом перехода и имеющий две формы представления. В первом случае это 10-битовое число со знаком, ис-

пользуемое как смещение команды относительно начала команды, содержащей адрес перехода. Во втором случае это 16-битовое число без знака, используемое как смещение относительно начала сегмента, содержащего текущую команду. Смещения по отношению к командам в пределах сегментов команд задаются в битах, а не в байтах (т. е. начало и конец любой команды располагаются на границах битов, а не байтов).

#### ТАБЛИЦА ОБЪЕКТОВ

Теперь целесообразно еще раз обратиться к рассмотренному таблице объектов, которая используется для преобразования дескрипторов доступа в физические адреса сегментов, и детально ознакомиться с ее содержимым. Таблица объектов содержит список 128-битовых дескрипторов. Для задания типа дескриптора используются два или пять младших двоичных разрядов. Форматы дескрипторов показаны на рис. 17.10. Рассмотрим

##### Дескриптор объекта

127

Законсервировано	Номер ячейки	Зарезервировано	Законсервировано	Длина сегмента	Адрес базы сегмента	0
Указатель "Сегмент не очищен"			Тип объекта/сегмента Класс процессора Повторный запрос		бит доступа бит изменения блокиратора вода - выбор указатель помехи базовый тип бит готовности	11

##### Заголовок таблицы объектов

Запрос помехи	Номер ячейки	Зарезервировано	Законсервировано	Номер ячейки, на которую обозначается ссылка	Индекс конца	Свободный индекс	Законсервировано	00 0 00
				Повторный запрос помехи		Зарезервировано		

##### Свободный дескриптор

Законсервировано	Зарезервировано	Законсервировано	00 1 00
— Повторный запрос			

##### Дескриптор типа

Дескриптор доступа для объекта расширенного типа	Номер ячейки	Зарезервировано	Законсервировано	Дескриптор доступа для объекта "отредактированный типа"	Законсервировано	01
				Повторный запрос	Общее/частковый бит готовности	

##### Дескриптор аффинажа

Дескриптор доступа для сегмента, содержащего аффинаж	Номер ячейки	Зарезервировано	Смещение базы	Длина аффинажа	Индекс страницчика	Индекс сезмента	10
			Тип объекта/сегмента Класс процессора Повторный запрос		базовый тип бит готовности		

Рис. 17.10. Форматы дескрипторов в таблице объектов. (\* — Имеет место обращение на промежуточной стадии реализации доступа.)

**Таблица 17.1.** Кодирование содержимого поля типа объекта/сегмента в дескрипторе объекта

Значение	Базовый тип	
	Доступ	Данные
00000	Сегмент доступа общего назначения	Сегмент данных общего назначения
00001	Зарезервировано	Стек операндов контекста
00010	Домен (корневой)	Таблица объектов
00011	Зарезервировано	Сегмент «команды»
00100	Контекст (корневой)	Сегмент данных контекста
00101	Процесс (корневой)	Сегмент данных процесса
00110	Процессор GDP (корневой)	Сегмент данных процессора
00111	Порт (корневой)	Сегмент данных порта
01000	Транспортер (корневой)	Сегмент данных транспортера
01001	Ресурсы памяти (корневые)	Сегмент данных ресурсов памяти
01010	Определение типа	Сегмент данных связи
01011	Зарезервировано	Сегмент данных управления дескриптором
01100	>	Сегмент данных управления аффинажем
01101	Зарезервировано	Зарезервировано
01110	>	>
01111	>	>

дескрипторы объекта (памяти) и заголовка таблицы объектов, а также так называемый свободный дескриптор. Основной информацией дескриптора объекта или памяти являются 24-битовый физический адрес базы (начала) сегмента, 16-битовая длина сегмента (в байтах) и его тип. Информация о типе сегмента размещается в двух полях: 1-битовом поле «базовый тип» (0 — сегмент данных, 1 — сегмент доступа) и 5-битовом поле «тип объекта/сегмента» (возможное содержимое этого поля дано в табл. 17.1). Бит готовности указывает, можно ли использовать данный дескриптор объекта для обращения к сегменту. Бит — указатель наличия соответствующей памяти, будучи равным 0, вызовет регистрацию ошибки, если при этом предпринимается попытка адресации посредством дескриптора объекта. Этот бит может использоваться программными средствами системы для индикации отсутствия текущего сегмента в памяти. Бит изменения обычно имеет значение 0, которое ему присваивается перед началом работы. Процессор присваивает этому биту значение 1 при любых изменениях запоминающей среды (памяти) в сегменте. Бит доступа также обычно имеет значение 0, устанавливаемое программными средствами перед началом работы или периодически. Процессор присваивает этому биту значение 1 в том случае, когда реализован доступ к

запоминающей среде этого сегмента. Перечисленные биты (поля формата дескриптора) предназначены для организации перемещения сегментов между различными уровнями иерархии адресного пространства памяти системы, т. е. для постстраничного обмена средствами операционной системы.

Номер уровня 16 бит указывает на «продолжительность жизни» сегмента. Если его значение равно 0, это подразумевает, что сегмент был выделен из общего пула («кучи») памяти<sup>1)</sup>. Если значение номера уровня отлично от нуля, это указывает на факт создания сегмента в процессе обращения к подпрограмме, причем это значение является номером динамически устанавливаемого уровня вложения активации подпрограммы. Эта информация совместно с флагом повторного запроса (памяти, занимаемой объектами) используется для процедуры «сбора мусора». Бит — указатель «сегмент не очищен» равен 1, если сегмент содержит биты, не равные 0. Поскольку все выделяемые для работы сегменты «очищены», пространство памяти, выделяемое из «неочищенного» сегмента заполняется двоичными нулями.

Все поля формата дескриптора объекта, помеченные как зарезервированные, аппаратными средствами системы не используются и предоставляются в распоряжение программных средств (например, операционной системе). Поля, помеченные как резервируемые, либо не используются, но могут быть использованы при расширении системы в будущем, либо используются аппаратными средствами для целей, не предусмотренных в определении архитектуры системы.

Теперь рассмотрим свободный дескриптор. Он просто описывает незанятую позицию в таблице объектов. Зарезервированные области могут использоваться программными средствами для соединения всех свободных дескрипторов вместе.

Дескриптор, именуемый «заголовок таблицы объектов», включает в себя некоторую общую информацию о таблице объектов. Поле свободного индекса содержит ссылку на первый дескриптор в таблице объектов, поле индекса конца — ссылку на последний дескриптор этой таблицы. В 32-битовом поле запроса памяти указывается общее количество памяти, которая может быть выделена для сегментов, связанных с данной таблицей объектов. Если во всех разрядах этого поля содержатся 1, то ограничения на запрашиваемую память отсутствуют. В противном случае указанные ограничения претерпевают изменения («подстраиваются») при каждом выделении памяти для сегментов или освобождении ранее выделенной памяти.

<sup>1)</sup> Слово «куча» по-английски *heap*; отсюда происхождение термина «хип-флаг» (*heap flag*). — Прим. ред.

Если таблицу объектов ассоциировать с некоторым процессом, то упомянутый запрос памяти в системе iAPX 432 можно считать подобным поятию АМ (имеющейся в наименовании памяти) в системе SWARD.

## ЗАЩИТА ОТ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА

Как отмечалось выше, предотвращение несанкционированного доступа в системе iAPX 432 основано на использовании защищенных потенциальных адресов, снабженных правами объекта (на чтение содержимого сегмента или запись информации в сегмент) и правами системы (на выполнение команд, определяющих сегментом как системным объектом). Защита потенциальных адресов осуществляется путем их записи в сегмент особого типа (сегмент доступа). В дополнение к этому система iAPX 432 располагает некоторыми средствами, по своим функциям родственными средствам защиты потенциальных адресов.

### ОБЪЕКТ «УПРАВЛЕНИЕ ДЕСКРИПТОРОМ»

Подобно системе SWARD, система iAPX 432 не содержит привилегированных команд. Однако в отличие от системы SWARD никаким программам рассматриваемой здесь системы не разрешается выполнять какую-либо команду односторонне. Некоторые команды требуют возможности обращения к объекту «управление дескриптором» (ОУД) как к операнду. В этом объекте указывается, что разрешается делать соответствующей команде.

Такой объект является 32-битовым сегментом данных. К нему могут обращаться следующие три команды: CREATE-TYPED-SEGMENT, RESTRICT-RIGHTS и AMPLIFY-RIGHTS. В ОУД имеются соответствующие поля. Во-первых, это 1-битовое поле «базовый тип» и 5-битовое поле «тип объекта/сегмента» (см. табл. 17.1); содержимое этих полей определяет тип сегмента, который может быть создан любым «владельцем» средств адресации к данному ОУД. Во-вторых, в ОУД имеются поля прав объекта и прав системы, определяемые согласно описаннию дескриптора доступа. В дополнение к этому дескриптор доступа к ОУД, как к объекту, имеет права системы. Последние представляют собой возможность использовать ОУД, во-первых, для создания системного объекта (т. е. любого объекта, кроме сегментов данных или общего доступа) и, во-вторых, для расширения прав в дескрипторе доступа.

Для создания сегмента системного типа (например, домена, корневого сегмента доступа процесса, сегмента данных процес-

са) используются команда CREATE-TYPED-SEGMENT, непосредственно формирующая сегмент, а затем команды манипуляции данными для инициализации этого сегмента. Чтобы предотвратить произвольное создание объектов какой-либо программой, команда CREATE-TYPED-SEGMENT должна иметь возможность обращаться к ОУД как к операнду, а ее дескриптор доступа должен иметь права системы на создание системного объекта. После этого на основании информации в ОУД определяется тип сегмента.

#### ОГРАНИЧЕНИЕ И РАСШИРЕНИЕ ПРАВ ДОСТУПА В ПОТЕНЦИАЛЬНЫХ АДРЕСАХ

В отличие от системы SWARD в системе iAPX 432 имеется возможность расширения прав доступа в потенциальных адресах. Очевидно, что при этом необходимо принять соответствующие меры по защите от несанкционированных действий. Для этой цели используется ОУД. Для объекта определенного типа соответствующий ОУД обычно находится под сильным контролем спецификатора типа этого объекта и используется для получения особых привилегий для этого объекта.

Команда AMPLIFY-RIGHTS, предназначенная для расширения упомянутых прав, позволяет обращаться к дескриптору доступа и ОУД. Дескриптор доступа к ОУД должен содержать права системы, что формулируется следующим образом: «может быть использован для расширения прав». ОУД содержит флаги прав объекта и системы, результаты выполнения логических операций ИЛИ над которыми записываются в соответствующие поля адресуемого дескриптора доступа.

Команда RESTRICT-RIGHTS, предназначенная для ограничения рассматриваемых прав доступа, позволяет обращаться к дескриптору доступа и ОУД или 32-битовой области, похожей на ОУД (называемой «забытой ОУД»). Содержащаяся в ОУД информация о правах объекта и системы подвергается логической операции И с записью результата в соответствующие поля адресуемого дескриптора доступа с целью ограничения его прав доступа и сохранения их неизменными. Наличие «забытого ОУД» позволяет любой программе односторонне ограничивать права доступа в любом из ее дескрипторов доступа.

Система iAPX 432 предоставляет возможность использования (по усмотрению программиста) одной дополнительной операции контроля процесса модификации прав доступа. По желанию пользователя можно присвоить значение 1 биту «Проверка типа» в ОУД. Тогда при любом использовании ОУД для изменения прав доступа система прежде всего проверяет, соответствует ли тип сегмента, к которому производят обращение под-

лежащий модификации дескриптор доступа, тому типу, который задан в ОУД (например, в распоряжении может быть ОУД, допускающий расширение прав только в тех дескрипторах доступа, которые адресуются к портам связи).

Возможность расширения прав доступа оказывается наиболее эффективным средством в сочетании с возможностями спецификатора типа объекта. Рассмотрим, например, такой спецификатор как часть операционной системы, в ведении которой находятся порты связи. Одной из функций такого спецификатора могло бы быть «создание порта», другой — «определение размера очереди запросов к порту». Реализация первой функции вероятнее всего сводилась бы к выдаче дескриптора доступа с правами системы на прием и передачу через порт, однако без прав объекта на анализ или модификацию сегментов порта. Выполнение операций над дескриптором доступа по реализации второй функции приводило бы к временному расширению прав доступа дескриптора на анализ состояния порта.

#### ОБЪЕКТЫ УПРАВЛЕНИЯ АФФИНАЖЕМ

Формирование аффинажа — процесс фрагментации сегмента — можно рассматривать, согласно изложенному в предыдущем разделе, как действия по созданию нового объекта или сегмента из смежных областей памяти существующего сегмента. Если для такого нового сегмента хотят сохранить те же самые базовый и системный типы, что и у существующего сегмента, то не требуется запроса никаких специальных привилегий; достаточно воспользоваться командой CREATE-GENERIC-REFINEMENT. По этой команде формируется дескриптор доступа, адресующийся не к дескриптору объекта, а к дескриптору аффинажа в таблице объектов (рис. 17.10). Последний используется для получения дескрипторов доступа к особым переменным программы, представляющим фактические параметры для подпрограммы.

В отдельных случаях может возникнуть необходимость в создании сегмента доступа с последующим выделением из него других сегментов; некоторые из них должны быть даже корневыми сегментами для системных объектов различного типа. Например, в таком случае при программировании на языке Ада задача могла бы быть представлена как объект «домен» (сегмент доступа), объект «процесс» (один сегмент доступа, один сегмент данных), один или несколько портов связи (один сегмент доступа, один сегмент данных) для каждого входа в задачу и один или несколько сегментов «команды» (сегменты данных). Вместо того чтобы для каждого подобного объекта создавать свои сегменты, появляется возможность сформировать

единственный сегмент доступа, пользуясь которым можно выделять сегменты доступа для различных объектов. Это могло бы привести к более эффективному управлению памятью. Так, при «перекачивании» сегментов из основной памяти на диски и обратно можно было бы избавиться от манипулирования большим количеством сегментов малых размеров; взаимосвязанные сегменты оказываются физически сгруппированными вместе, подобно группе доступа в архитектуре Системы 38 фирмы IBM.

Для получения описанных возможностей необходимо наличие специальных привилегий, заключенных в объекте «управление аффиажем». Используемая для этих целей команда CREATE-TYPED-REFINEMENT обуславливает обращение с этим объектом как с операндом. Информация, заключенная в 32-битовом сегменте данных объекта «управления аффиажем», указывает системный тип объекта, который может быть создан как аффиаж.

Кроме перечисленных выше возможностей время от времени может потребоваться функция, состоящая в следующем: посредством заданного дескриптора доступа к аффиажу создать дескриптор доступа к сегменту, охватывающему (содержащему) данный аффиаж. Такая операция должна быть защищена от несанкционированного доступа, поскольку в противном случае возможно нарушение принципа защиты с помощью доменов. Выполняющая эту функцию команда RETRIEVE-REFINED-ОБЪЕКТ предписывает использование объекта «управление аффиажем» в качестве операнда.

Другие вопросы, связанные с проблемой защиты от несанкционированного доступа, рассматриваются в разделе, содержащем описание объектов, определяемых программными средствами.

## СТРУКТУРА ПРОГРАММЫ

Для описания структуры программ системы iAPX 432 используются объекты «домен», «команды» и «коинтекс». Объекты первых двух типов позволяют описывать статическую структуру программы, объекты третьего типа — динамическую, так как создаются машиной в результате вызова подпрограмм.

Объект «команды» содержит последовательность машинных команд, относящихся обычно к однородной подпрограмме. Поскольку для адресации к командам используется смещение, выражаемое в битах, размер объектов «команды» не превышает 8192 байт. Подпрограммы, размер которых превышает указанный предел для объектов «команды», могут быть представлены посредством группы таких объектов.

Каждый объект «команды» — это один сегмент данных, содержащий семь 2-байтовых полей, за которым следуют команды. Эти поля содержат следующую информацию: 1) длину сегмента доступа контекста (эта величина используется машиной с целью выделения корневого сегмента доступа для объекта «контекст» при вызове объекта «команды»); 2) длину сегмента данных контекста; 3) длину стека операндов контекста; 4) смещение первой команды, подлежащей выполнению; 5) индекс ДД к сегменту данных — констант в объекте «домен»; 6) индекс ДД к сегменту ошибок объекта в объекте «домен» (команды этого сегмента подлежат выполнению при обнаружении ошибки в контексте); 7) индекс ДД к сегменту трассировки объекта в объекте «домен» (команды этого сегмента подлежат выполнению, когда осуществляется выполнение трассируемой команды).

В ДД к сегменту команд указываются следующие системные права: 1) можно ли вызывать сегмент команд; 2) можно ли выполнять трассировку процесса выполнения команд сегмента.

Объект «домен» служит для группировки набора объектов «команды» и сегментов данных. Понятию «домен» системы iAPX 432 соответствует понятие «пакет» в программе на языке Ада. При выполнении своих функций домен адресуется к объектам «команды», которым в языке Ада соответствуют подпрограммы и другие конструкции из операторов описания, входящие в состав пакета, а также обращается к сегментам данных, описывающим статические переменные общего и частного доступа, а также константы, генерируемые компилятором.

Объект «домен» — это одиночный сегмент доступа. В отличие от большинства объектов других типов он не имеет фиксированного формата. Порядок следования в нем дескрипторов доступа к объектам «команды», сегментам данных и другим адресуемым элементам определяется программными средствами, например компилятором. Путем создания аффинажа домена некоторые из связанных с ним объектов (подмножество сегментов команд и данных) можно сделать объектами общего доступа (т. е. адресуемыми из других доменов), а другие сохранять как объекты частного доступа (т. е. адресуемые только из тех объектов «команды», которые взаимосвязаны с данным доменом). Архитектурой системы iAPX 432 не предусмотрено определение конкретного значения системных прав в дескрипторе доступа.

Объект «контекст» играет ключевую роль в системе iAPX 432, поскольку определяет динамическое состояние «окружения» программы во время ее выполнения. Это единственный объект, создаваемый машиной иначенным образом. Выполнение програм-

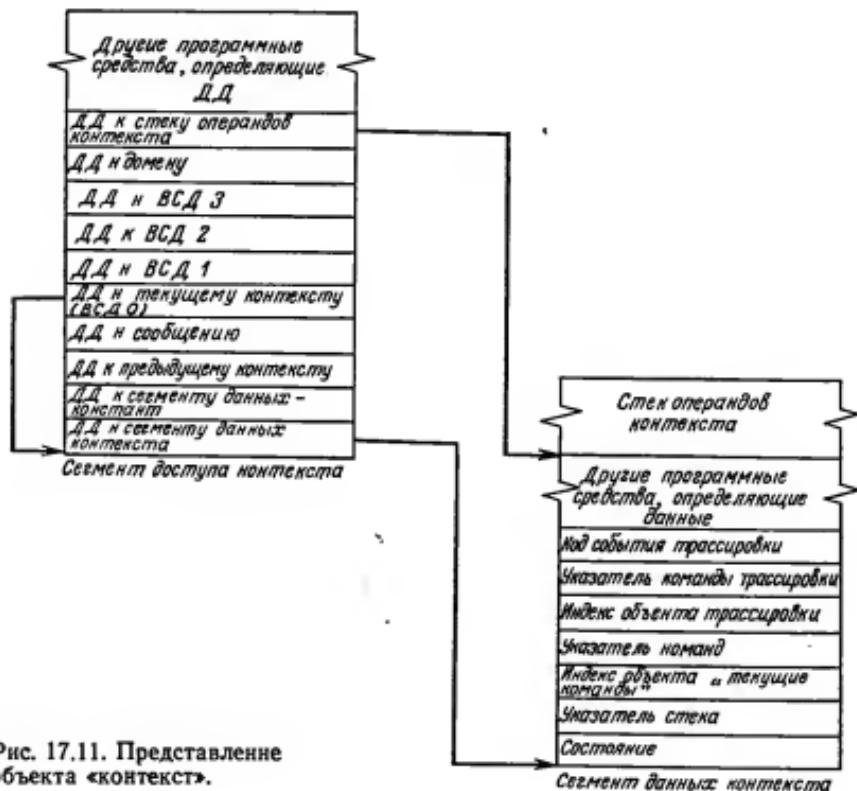


Рис. 17.11. Представление объекта «контекст».

мы представлено в системе списком контекстов — одним для каждой активной подпрограммы. При этом последним в списке является контекст, описывающий динамику всех средств системы, «окружающих» процесс во время его протекания.

При обращении к сегменту команд машина создает два сегмента: корневой сегмент доступа и один сегмент данных (рис. 17.11). Информация из предыдущего контекста и вызванного сегмента команд используется для инициализации новых сегментов контекстов. Если сегмент команд был вызван командой CALL-CONTEXT-WITH-MESSAGE (а не просто командой CALL), дескриптор доступа сообщения (обычно сегмент доступа, адресующийся к параметрам) вставляется в новый сегмент доступа. Следует помнить, что сам сегмент доступа контекста адресуется селектором 0 входного сегмента доступа. Он также содержит дескрипторы доступа к ВСД, выбираемым селекторами 1, 2 и 3.

Сегмент данных контекста описывает текущее состояние контекста. Так называемое поле состояния сегмента позволяет

управлять точностью и процедурой округления операций с плавающей точкой в контексте. Содержимое этого поля может модифицироваться посредством команды SET-CONTEXT-MODE. Указатель стека определяет положение вершины стека операндов в каждый данный момент. Индекс объекта «команды» указывает местоположение индекса дескриптора доступа сегмента текущих команд в пределах соответствующего домена. Указатель команды определяет положение текущей машинной команды. Во время выполнения процессором контекста предсказать содержимое большинства перечисленных полей невозможно.

Другая, следующая за описанными полями часть сегмента данных контекста используется компилятором в качестве памяти для локальных переменных подпрограммы. И наконец, часть этого сегмента, имеющая адреса с наибольшими значениями, служит стеком операндов.

Согласно изложенному в разделе об адресации, в качестве данных, адресуемых операндами команды, может использоваться только содержимое стека операндов и сегмента, дескриптор доступа которого находится либо в корневом сегменте доступа контекста (поскольку этот сегмент является ВСД 0), либо в сегменте доступа, адресуемом дескрипторами доступа к ВСД 1, ВСД 2 или ВСД 3 в корневом сегменте контекста. Не исключается возможность для компилятора использовать ВСД 0 в качестве средства доступа к локальным переменным (в сегменте данных контекста) и константам (которые могут быть помещены в сегмент данных, связанный с данным объектом «домен»). Объект, посылаемый этому контексту как сообщение, может иметь его дескриптор доступа, скопированный в поле ВСД 1, что дает командам возможность доступа к параметрам путем выбора ВСД 1 в адресах операндов. ВСД 2 можно использовать для обращения либо к сегменту доступа, адресующемуся к статическим переменным, связанным с данным доменом, либо к глобальным переменным (переменным общего доступа), связанным с процессом.

Если программа выполняет обработку списка какого-либо типа, компилятор может резервировать ВСД 3 для этой цели. Например, если сегмент А производит обращение к сегменту В, который в свою очередь производит обращение к сегменту С, работу можно начать, располагая дескриптором доступа к А в предназначенному для ВСД 3 элементе сегмента доступа контекста. Для получения возможности адресации к дескрипторам доступа сегмента В необходимо дескриптор доступа к В поместить в элемент, предназначенный для ВСД. Это можно сделать путем выполнения команды ENTER-ACCESS-SEGMENT, задавая в качестве операндов величину 3 для ВСД 3 и значение дескриптора доступа в А к В.

## ПРОЦЕССОРЫ И ПРОЦЕССЫ

Архитектуре системы iAPX 432 присущи хорошая организация одновременно протекающих процессов и высокая степень согласования совместной работы группы процессоров. Прежде всего заслуживают внимания средства синхронизации и связи как между процессорами, так и между процессами, а также средства, дающие возможность программному обеспечению (например, операционной системе) принимать участие в планировании протекания процессов.

### ОБЪЕКТ «ПРОЦЕССОР»

Объект «процессор» используется при описании архитектуры системы для представления реального процессора. (В дальнейшем, однако, рассмотрению подлежит только процессор общего назначения.) Обычно объект «процессор» может находиться в одном из следующих двух состояний: 1) имеется связанный с ним объект «процесс», т. е. процессор выполняет команды согласно некоторому контексту в этом процессе; 2) процессор находится в очереди на обслуживание некоторым портом диспетчеризации, т. е. процессор ожидает получения работы. На рис. 17.12 дано схематическое изображение объекта «процессор». Он представляет собой некоторую структуру данных, состоящую из корневого сегмента доступа, двух сегментов данных и объекта «траиспортер» (состоящего из сегмента доступа и сегмента данных). (Сегментом данных глобальных связей обычно пользуются все процессоры.) Сегменты данных для локальных и глобальных связей используются для обмена сообщениями между процессорами; ниже более подробно будет описано назначение этих сегментов.

Сегмент доступа процессора содержит два важных средства адресации: дескриптор доступа к справочнику таблиц объектов и дескриптор доступа к нормальному порту. Первый дескриптор обеспечивает процессору доступ к справочнику таблиц объектов с целью разрешения адресных ссылок, второй адресует его к порту диспетчеризации, из которого процессор получает процессы. Большая часть прочей информации сегмента доступа процессора используется в случае возникновения аномальных ситуаций. Аварийный порт — это специальный порт диспетчеризации, к которому процессор подключается при обнаружении в машине не поддающейся восстановлению потери питания. Порт диагностики — еще один специальный порт диспетчеризации, к которому подсоединяется процессор при обнаружении в выполняемой программе ошибок некоторого типа. Порт реконфигурации системы используется для подключения к нему процес-

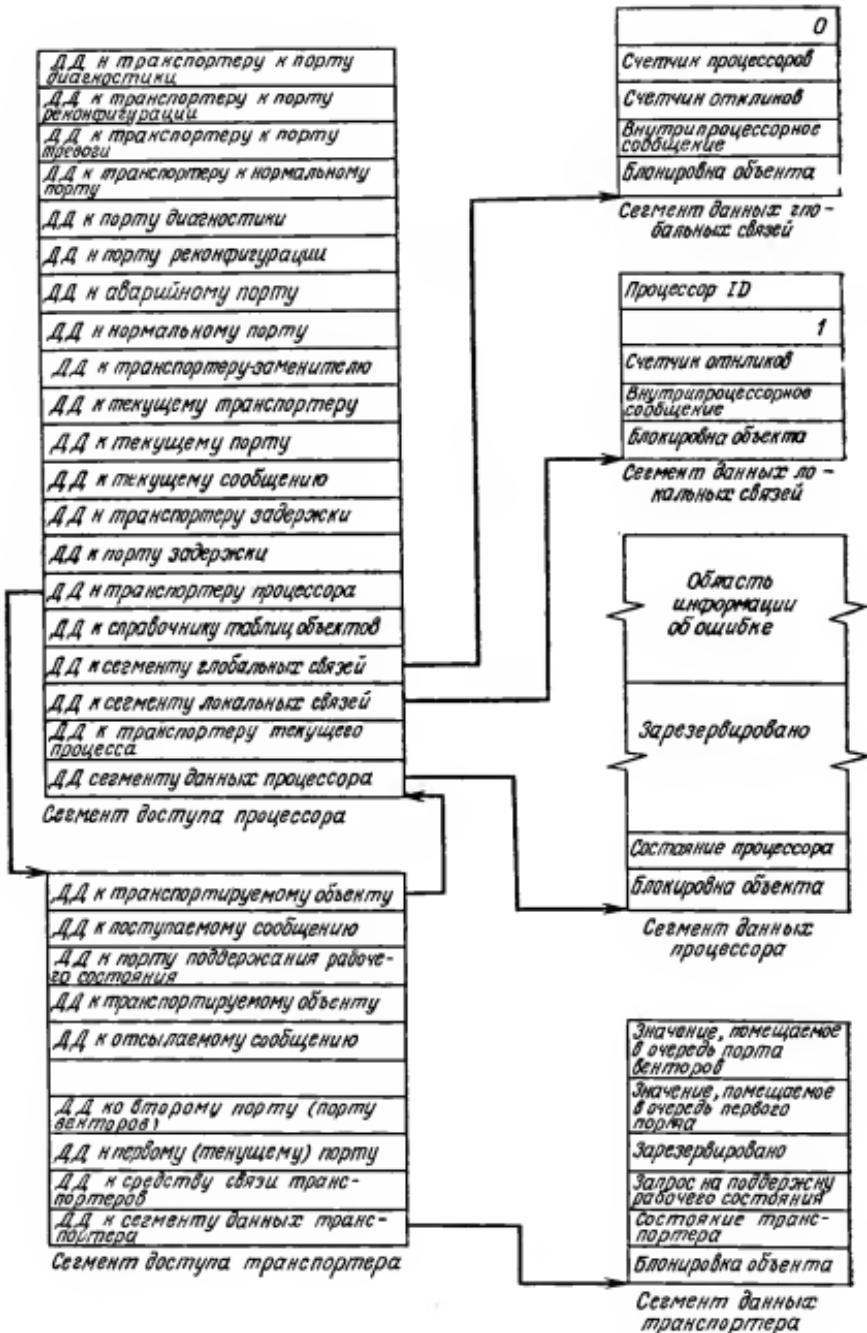


Рис. 17.12. Представление объекта «процессор».

сопа в тех случаях, когда возникают особые ситуации в работе памяти совместно с общей шиной, связанные с решением задач возможного будущего расширения конфигурации аппаратных средств системы.

Сегмент данных процессора содержит информацию о состоянии процессора и некоторых ошибках (см. раздел по обработке ошибок). Информация включает следующие сведения: 1) текущее состояние процессора (бездействие, начальная установка, выполнение или выбор процесса); 2) режим диспетчеризации (нормальный, аварийный, диагностический или реконфигурации); 3) установлен ли данный процессор другим процессором в состояние «Останов»; 4) приняты ли сообщения, передаваемые другими процессорами; 5) значение 8-битового идентификатора процессора. Во время выполнения процесса регистрация точного состояния процессора в соответствующем поле его сегмента данных не производится. Информацию об истинном состоянии процессора в интервале времени его работы можно получить в программе с помощью команды READ-PROCESSOR-STATUS-AND-CLOCK.

#### ОБЪЕКТ «ПРОЦЕСС»

На рис. 17.13 изображен объект «процесс». Он состоит из корневого сегмента доступа, сегмента данных, определяемого программными средствами сегмента доступа, а также сегмента доступа и связанного с ним сегмента данных, объединенных под общим названием «траиспортер» и описываемых в разделе, посвященном коммуникациям между процессами. Отметим, что процесс адресуется к таблице объектов. Как уже указывалось, весьма вероятно, что созданный программными средствами процесс будет располагать своей собственной таблицей объектов, на основании которой можно создавать дескрипторы объектов, подобно тому как процесс создает сегменты.

Процесс адресуется и к порту диспетчеризации. Поместить процесс в порт диспетчеризации — это функция аппаратных, а не программных средств машины. Например, если процессор принимает решение приостановить выполнение своего собственного процесса и выбрать из своего порта диспетчеризации другой процесс, то такому процессору необходимо поместить текущий процесс в очередь на обслуживание портом диспетчеризации. Порт планирования — это порт связи, которому процессор посыпает процесс после того, как последний находился на обслуживании в порту диспетчеризации заданное число раз. Порт ошибки — это порт связи, в очередь на обслуживание которым помещается процесс в случае обнаружения ошибок в самом процессе.

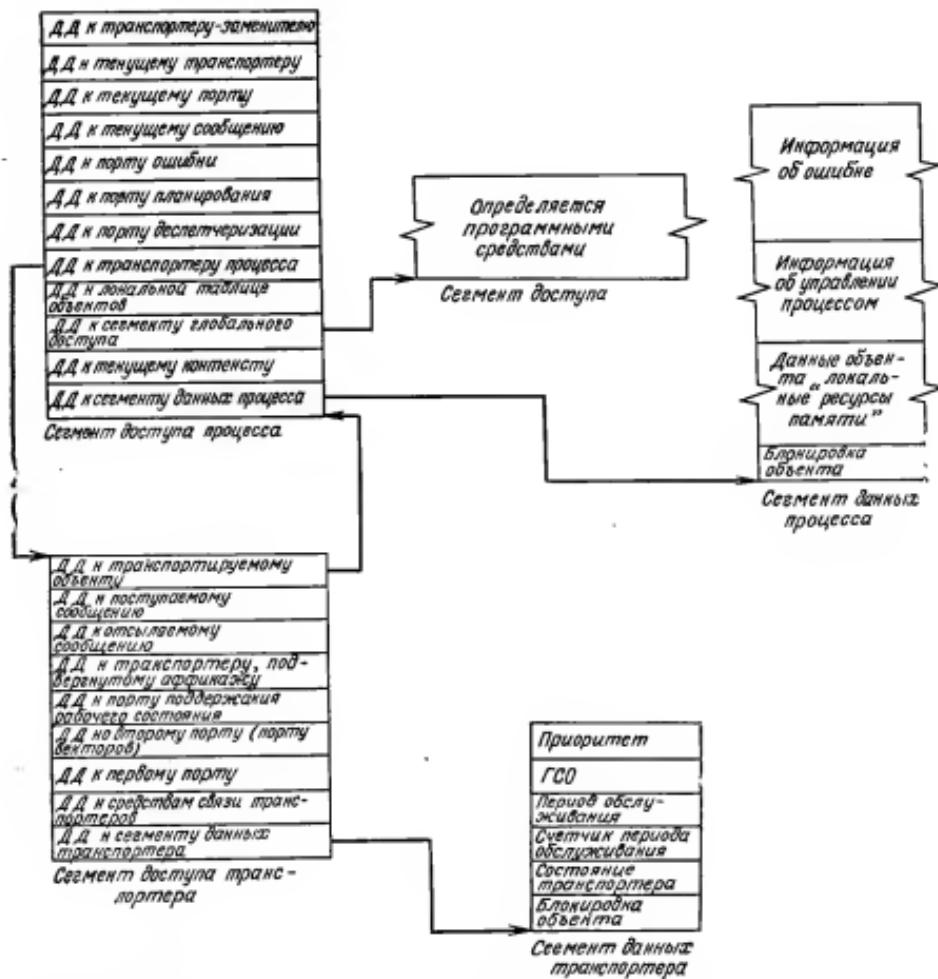


Рис. 17.13. Представление объекта «процесс».

Сегмент данных процесса содержит информацию, необходимую для управления памятью и обработки ошибок, а также так называемую управляющую информацию, формат представления которой показан на рис. 17.14. Период обслуживания и число периодов обслуживания — это параметры планирования обслуживания процессов процессорами. Посредством этих параметров программные средства системы (например, операционная система) могут оказывать влияние на диспетчеризацию процессов процессорами. Период обслуживания — это интервал времени (интервал выполнения), при превышении которого процес-

сор вынужден поместить процесс обратно в порт диспетчеризации и попытаться выбрать другой процесс. Число периодов обслуживания указывает количество периодов обслуживания, предоставленных процессу; при попытке запроса большего числа периодов обслуживания процессор посыпает процесс в порт диспетчеризации.

Находящаяся в сегменте данных процесса информация о состоянии процесса содержит следующие данные: 1) связан ли в настоящее время процесс с каким-либо процессором (т. е. выполняется ли процесс); 2) обнаружена ли ошибка в процессе; 3) ожидает ли процесс сообщения, находясь у некоторого порта; 4) подлежат ли ошибки процесса обработке как таковые или как ошибки контекста; 5) какой режим трассировки задач (никакой, трассировка ошибок, трассировка передачи управления, полная трассировка). Информация о продолжительности процесса представлена 32-битовой величиной, указывающей (в единицах времени, соответствующих длительности периода вида тогового генератора) время, которое уже израсходовано на обслуживание данного процесса. Номера уровней ВСД — это номера уровней контекстов, в которых были созданы входные сегменты доступа текущего контекста.

На рис. 17.13 показан еще один сегмент доступа, связанный с сегментом доступа процесса. Его функциональное назначение в системе не является раз и всегда определенным; предполагается, что он может быть использован для определения местоположения глобальных объектов или переменных процесса. Поскольку обычно процесс не имеет дескриптора доступа к самому себе, в системе предусмотрена специальная команда ENTER-GLOBAL-ACCESS-SEGMENT, позволяющая иметь дескриптор доступа к этому сегменту, который вставляется в одно из полей для ВСД в контексте.

#### ПЛАНИРОВАНИЕ И ДИСПЕТЧЕРИЗАЦИЯ ПРОЦЕССОВ

Система iAPX 432 располагает чрезвычайно гибкими средствами программного управления планированием и диспетчеризацией процессов. В дальнейших рассуждениях не будем прово-

Зарезервировано
Номер уровня ВСД3
Номер уровня ВСД2
Номер уровня ВСД1
Зарезервировано
Процесс ID
Период действия процесса
Состояние процесса
Период обслуживания
Счетчик периодов действия процесса
Информация об управлении процессом

Рис. 17.14. Формат представления информации об управлении процессом в сегменте данных процесса.

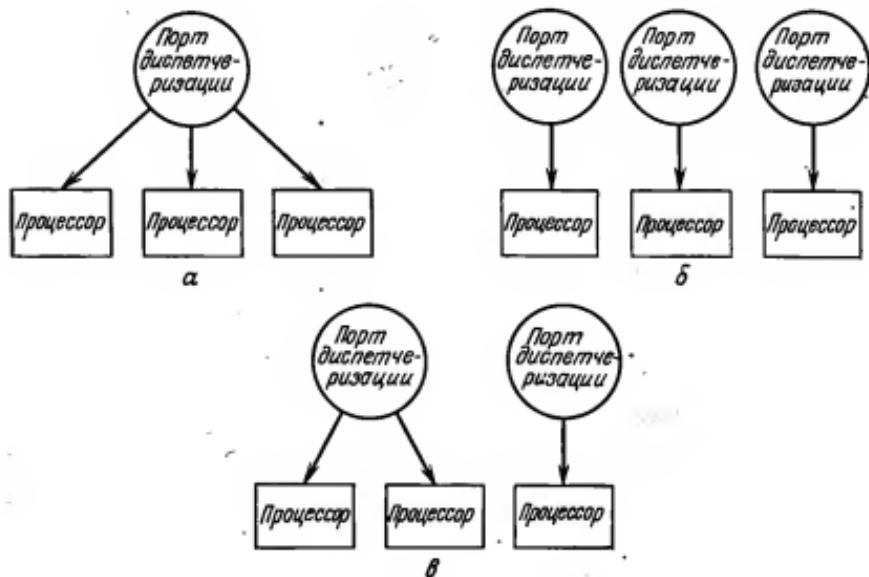


Рис. 17.15. Три возможных варианта диспетчирования.

дить различий между этими средствами и операционной системой, поскольку операционной системе не присущи особые права, функции и привилегии, т. е. и некоторая прикладная программа может, например, создавать процессы и управлять ими самостоятельно, получив необходимые исходные данные, такие, как объект «управление дескриптором», позволяющий создавать объекты «процесс». Упомянутая гибкость средств достигается благодаря 1) развитой системе взаимосвязей между портами диспетчирования и процессорами, 2) присваиванию портам диспетчирования определенных правил организации очередей, 3) присваиванию процессам параметров плана их обслуживания, 4) наличию порта планирования и 5) наличию порта задержки.

Развитая система взаимосвязей между портами диспетчирования и процессорами позволяет с использованием средств операционной системы создавать многочисленные конфигурации схем диспетчирования процессов. На рис. 17.15 показаны три такие схемы. В схеме на рис. 17.15, а все процессоры пользуются общим пулом работ (процессов) в единственном порте диспетчирования. Следовательно, в этом случае операционная система не вовлечена в распределение процессов между процессорами. Схема, показанная на рис. 17.15, б, является противоположным случаем. Здесь у каждого процессора свой порт диспет-

черизации, и операционной системе приходится принимать решение, между какими процессами и процессорами устанавливать связи при создании процессов. Схема на рис. 17.15, в занимает некоторое промежуточное положение между двумя рассмотренными экстремальными случаями: пул процессоров связан с одним портом диспетчеризации, а другой подобный порт предоставлен некоторому процессору (например, для обслуживания какого-либо процесса с экстраординарными характеристиками).

Порт диспетчеризации — это объект, представляющий очередь процессов или процессоров (но не одновременно и тех и других). Формат описания такого порта показан на рис. 17.2. Хотя управление порядком обслуживания очередей в порте диспетчеризации может быть предоставлено операционной системе, столь высокий уровень управления программными средствами обычно не допускается, поскольку зачастую он может оказаться неэффективным. Вместо этого операционную систему можно использовать для определения системы правил организации очередей для каждого порта диспетчеризации и для задания параметров для каждого процесса. Следовательно, в системе iAPX 432 принятая следующая стратегия планирования обслуживания процессов: аппаратные средства ответствены за планирование на низком уровне, операционная система — на высоком.

Поле состояния порта диспетчеризации содержит информацию о том, какое правило организации очередей задано для данного порта: «поступивший первым — обрабатывается первым» (FIFO) или «приоритет и гарантированные сроки обслуживания (ГСО)». Эта информация используется процессором всякий раз, когда необходимо поместить процесс в очередь на обслуживание портом диспетчеризации (например, по истечении периода времени обслуживания процесса процессором). Если очередь организована по принципу FIFO, то процесс просто помещается в ее конец. При организации очереди с учетом приоритетов процессы располагаются в соответствии со значениями приоритетов. Процессы с равными значениями приоритетов обслуживаются согласно правилу ГСО. Каждому процессу предоставляется право указать время относительной задержки, т. е. максимальный интервал времени, в течение которого процесс может ожидать обслуживания, находясь в очереди. Перед помещением процесса в очередь процессор добавляет указанное время задержки к интервалу времени обслуживания процессором процесса. В результате получается так называемое предельное время запуска. (Часы отдельных процессоров синхронизированы между собой.) В соответствии со значением предельного времени запуска процессор помещает процесс в ту или иную пози-

цию очереди, организованной согласно порядку следования значений этих времеи.

Отметим, что некоторая дополнительная информация, такая, как приоритет процессов, поступает из объекта «транспортер», показанного на рис. 17.13. Эти объекты используются прежде всего для формирования очередей; они «транспортируют» некоторый объект в очередь. Выше отмечалось, что у портов имеется некоторое, фиксированное программными средствами пространство для размещения элементов очередей. Использование транспортеров дает возможность неограниченно расширять это пространство, формируя очередь в виде списка взаимосвязанных транспортеров. Объект «транспортер» создается для каждого объекта «процесс» и объекта «процессор», а также и для других целей (описываемых в последующих разделах).

В дополнение к упомянутым параметрам приоритета и задержки каждому процессу назначается период обслуживания (максимальный интервал времени выполнения процесса, по истечении которого процессор пытается переключиться на обслуживание другого процесса) и число периодов обслуживания (максимально допустимое число периодов обслуживания процесса процессором). При превышении значения последнего процессор выполняет неявно заданную команду SEND с целью передачи данного объекта «процесс» как сообщения заранее указанному порту планирования. Операционная система получает возможность принимать сообщения из этого порта. Она формирует решение о завершении процесса или перепланировке его обслуживания портом диспетчеризации, возможно, с измененными параметрами планирования обслуживания.

Процесс может оказывать влияние на планирование своего собственного обслуживания путем выполнения команды DELAY. Эта команда позволяет задать минимальный интервал времени, на который желательно приостановить выполнение процесса, прежде чем продолжить его дальнейшее выполнение. Следствием реализации команды DELAY является выполнение неявно заданной команды SEND, направляющей процесс на обслуживание в порт задержки, который относится к классу портов, отличному от ранее рассмотренных. Очередь «задерживаемых» процессов в порте задержки регулируется, согласно правилам ГСО, на основе желательных для процессов интервалов задержки их выполнения. В процессоре аппаратным путем реализован алгоритм, обеспечивающий периодическую инспекцию порта задержки, соответствующего данному процессору. Так, если процессор бездействует, ожидая поступления запроса на обслуживание у порта диспетчеризации, то, благодаря указанному алгоритму, он периодически «про-

буждает» себя и опрашивает порт задержки. По истечении интервала задержки, устанавливаемого упомянутыми правилами задержки выполнения процессов, процессор помещает процесс в очередь своего порта диспетчеризации.

#### ОБМЕН СООБЩЕНИЯМИ МЕЖДУ ПРОЦЕССОРАМИ

Архитектурой системы iAPX 432 предусмотрена для физически реализованных процессоров возможность взаимосвязи путем обмена сообщениями (эту возможность не следует путать с коммуникациями между процессами). Межпроцессорные связи иногда выполняются неявно (в результате происшествия в системе определенных событий, не поддающихся управлению программными средствами), а иногда этими связями управляют явным образом с помощью команд SEND-TO-PROCESSOR и BROADCAST-TO-PROCESSOR.

При реализации межпроцессорных связей необходимо использование сегментов данных локальных и глобальных связей, показанных на рис. 17.12. Имеются две формы представления межпроцессорных сообщений: одна — для передачи сообщений только специально указанным процессорам, другая — для передачи всем процессорам, которые совместно используют сегмент данных глобальных связей. Различают сообщения следующих типов:

Пуск процессора	Останов процессора
Задать режим приема глобальных сообщений	Отменить режим приема глобальных сообщений
Войти в режим «тревога»	Войти в режим диагностики
Войти в нормальный режим	Войти в режим реконфигурации
Сделать недействительной информацию в кэш-памяти сегмента данных	Сделать недействительной информацию в кэш-памяти таблицы объектов
Приостановить выполнение и установить информацию контекста	Приостановить выполнение и установить заново информацию процесса
Приостановить работу и установить информацию процессора	Приостановить работу и установить заново всю информацию процессора
Побудка бездействующего процессора	

Дадим краткие пояснения перечисленным директивам. Назначение пуска и останова процессора очевидно. Сообщение о задании режима приема глобальных сообщений выражает «желание» источника этого сообщения принимать глобальные

(предназначенные для всех процессоров) директивы. Сообщения о необходимости входа в тот или иной режим работы вынуждают процессор (процессоры) приостановить обслуживание текущего процесса и предоставить последний в распоряжение соответствующего порта диспетчеризации. Как указывалось, все процессоры снабжены соответствующей кэш-памятью, содержащей адресную информацию; для того чтобы сделать эту информацию недействительной, используются соответствующие директивы.

В процессе выполнения команд процессоры постоянно обновляют («поддерживают») информацию о текущем контексте, текущем процессе и процессоре как объекте архитектуры системы. Сообщения типа «приостановить работу и установить заново информацию процессора» заставляют процессор или процессоры обновить некоторую часть указанной информации, обращаясь за необходимыми данными к объектам в памяти. Подобные сообщения — директивы — организованы в виде некоторой иерархической структуры (например, директивы об обновлении информации процессора влечет за собой модификацию процессором информации контекста и процесса) и в большинстве случаев делают недействительным содержимое соответствующей кэш-памяти.

Каждое сообщение, передаваемое через шину пакетов памяти, состоит из кода — директивы и идентификатора процессора — источника сообщения. Подтверждение о том, что переданное межпроцессорное сообщение принято, регистрируется. Так, в соответствии с рис. 17.12, чтобы передать глобальное сообщение, процессор прежде всего копирует содержимое поля «счетчик числа процессоров» в поле «счетчик откликов» сегмента данных для глобальных связей. Каждый процессор подтверждает прием сообщения, уменьшая на 1 содержимое поля «счетчик откликов». Для сообщений, посыпаемых только одному процессору, все выполняется аналогичным образом, за исключением того, что содержимому счетчика откликов сегмента данных для локальных связей первоначально присваивается значение, равное 1.

## КОММУНИКАЦИИ МЕЖДУ ПРОЦЕССАМИ И ИХ СИНХРОНИЗАЦИЯ

В системе iAPX 432 имеется несколько возможностей для перемещения данных и их совместного использования процессами: средства передачи/приема (использующие порты связи), средства блокировки объектов, а также так называемые неделимые команды чтения/записи.

## ПОРТ СВЯЗИ

Порт связи представляет собой пример возможной реализации объекта «порт» (см. рис. 17.2). Порт связи (называемый в дальнейшем для краткости просто портом) используется для обмена сообщениями между процессами. Роль сообщения может выполнять одиночный объект любого типа. Так, сообщением может быть сегмент доступа или данных общего назначения, домен, процесс, процессор и т. д. Из системных прав, имеющихся в дескрипторе доступа, к порту относятся следующие: возможность передачи сообщения порту и возможность приема сообщений из порта.

Согласно архитектуре системы iAPX 432, тот, кто создает некоторый порт, ответствен за выделение фиксированного объема памяти для очередей как в сегменте данных порта, так и в его сегменте доступа. Для каждого элемента, поступающего в очередь порта, дескриптор доступа в сегменте доступа этого порта содержит указатель на этот элемент, а 8-байтовый элемент в сегменте данных описывает индекс следующего элемента этого сегмента, параметры приоритета или гарантированного срока обслуживания. Порты связи и диспетчеризации — это разновидности объекта «порт», и поэтому на них распространяется действие правил организации и обслуживания очередей. Так же как и рассмотренные выше порты диспетчеризации, каждый порт связи может быть создан с указанием одного из двух правил организации очередей: «поступивший первым — обрабатывается первым» или «приоритет — ГСО». Параметры планирования передачи процессов используются для определения порядка передачи/приема сообщений.

Для работы с портами служат две основные команды — SEND и RECEIVE. Команда SEND имеет два операнда: дескриптор доступа для порта и дескриптор доступа для объекта, подлежащего передаче в качестве сообщения. Если в порте в пространстве, предоставленном для очереди, имеется место, то дескриптор доступа для сообщения вставляется в сегмент доступа порта, параметры планирования посылающего процесса — в элемент очереди сегмента данных порта, а очередь подвергается реорганизации, если это необходимо. На этой стадии работы процесс не блокируется; его выполнение продолжается. Если же пространство, предоставляемое очереди, заполнено (или не существует), то выполняются следующие действия. Транспортер процесса помещается в конец списка транспортеров (начало и конец которого находятся в сегменте доступа порта). Он остается в этом списке до тех пор, пока не появится место в пространстве, предназначенном для очереди. На этой стадии посылающий процесс блокируется до тех пор, пока не появится место в упомянутом пространстве или — в слу-

чае отсутствия самого пространства — пока другой процесс не осуществит прием из данного порта. (Когда блокировка снимается, процессор автоматически посыпает процесс в соответствующий порт диспетчеризации. Если этот порт пуст, а некоторый процессор находится в очереди к нему, ожидая работы, то процессор посыпает сообщение упомянутому процессору о побудке бездействующего процессора.) При любом удалении какого-либо объекта из очереди порта процессор проверяет список транспортеров, упорядочиваемых по мере их прибытия извне. Если список не пуст, то процессор удаляет из него первый транспортер и помещает транспортируемый объект в очередь порта в соответствии с правилами организации очередей.

Команда RECEIVE содержит один операнд — дескриптор доступа порта. Если сообщение попадает в очередь порта, его дескриптор доступа удаляется и помещается в сегмент доступа транспортера принимающего процесса. Если сообщений нет, транспортер процесса используется для помещения этого процесса в очередь порта согласно правилу обслуживания «поступивший первым — обрабатывается первым», после чего процесс блокируется.

Для работы с портом связи используются две дополнительные команды (CONDITIONAL-SEND и CONDITIONAL-RECEIVE), каждая из которых располагает дополнительным операндом — логической (символьной) величиной. Первая из этих команд присваивает этой логической величине значение «ложно», если очередь порта заполнена и, следовательно, требуется поместить транспортер в очередь и заблокировать процесс. Вторая из названных команд действует подобно команде RECEIVE, если на входе порта имеется сообщение; в противном случае процесс не блокируется, а указанной логической величине присваивается значение «ложно».

#### ЗАМЕНИТЕЛИ СРЕДСТВ ПЕРЕДАЧИ/ПРИЕМА

Язык Ада располагает возможностями, обычно не предоставляемыми средствами передачи сообщений: принимать сообщения одновременно из нескольких портов. Например, согласно фрагменту программы на языке Ада,

```
select
  accept READ1 (R: In MASTER-RECORD) do
    .
    .
    end READ1;
  or
  accept READ2 (P: out UPDATE-RECORD) do
    .
    .
    end READ2;
end select;
```

задача должна ждать появления обоих входных элементов READ1 и READ2 (которые могли бы быть представлены как порты связи). Для решения подобной проблемы в системе iAPX 432 предусмотрено использование так называемого транспортера-заменителя (*surrogate carrier*)<sup>1)</sup>. В данном случае потребовалось бы два таких транспортера в дополнение к «обычному» транспортеру, связанныму с объектом «процесс».

Команда SURROGATE-RECEIVE помещает в очередь порта не принимающий процесс, а принимающий транспортер-заменитель. Последний содержит дескриптор доступа к процессу и дескриптор доступа к вторичному порту. Когда что-либо посыпается в первый порт и процессор узнает, что принимающей стороной является не процесс, а транспортер-заменитель, то этот процессор отсылает последний к вторичному порту. Чтобы иметь возможность ожидать сообщения из двух портов А и В одновременно, процесс инициирует транспортеры-заменители для указанных портов с назначением для этих транспортеров некоторого порта С в качестве вторичного порта. После этого инициируется обычный прием сообщений портом С и процессор ждет, когда будет получена информация из соответствующего транспортера-заменителя. Эта информация содержит дескриптор доступа к передаваемому сообщению.

Команда SURROGATE-SEND посылает сообщение при любых условиях без блокировки процесса, используя транспортер-заменитель в качестве транспортера сообщения. Когда порт связи располагает достаточным местом для сообщения, транспортер автоматически пересыпается к соответствующему пункту назначения (порту).

#### БЛОКИРОВАНИЕ ОБЪЕКТОВ

Из описания некоторых объектов, рассмотренных в предыдущих разделах, следует, что в системе iAPX 432 имеется возможность блокирования объектов. Для этой цели используются первые 16 бит следующих сегментов данных: процессора, транспортера процессора, локальных связей, глобальных связей, процесса, транспортера процесса, транспортера-заменителя, порта (любого типа). Шестнадцатибитовое поле для блокирования

<sup>1)</sup> Объект «транспортер-заменитель» подобен по структуре объекту «транспортер процесса». Различия между ними заключаются в следующем: 1) транспортер-заменитель может содержать информацию о планировании передачи и приема сообщений; транспортер процесса на это права не имеет; 2) каждому процессу соответствует один объект «транспортер процесса»; объект «транспортер-заменитель» не обязательно автоматически связан с процессами, в то же время одному процессу может принадлежать несколько таких объектов. — Прим. ред.

объектов имеется и в сегменте данных ресурсов памяти. Кроме того, программам предоставляется возможность использовать любую область памяти соответствующего размера в других сегментах данных для записи информации о том, заблокирован данный объект в настоящий момент времени или нет. Необходимо подчеркнуть, что блокирование и деблокирование объектов осуществляется как программными, так и аппаратными средствами системы. Программными средствами блокирования и деблокирования объектов являются команды LOCK-OBJECT и UNLOCK-OBJECT соответственно. Во многих случаях аппаратные средства процессора используют те же логические схемы, которые реализуют указанные команды.

Необходимость в синхронизации одновременно протекающих процессов не вызывает сомнений, но необходимость в синхронизации работы аппаратных средств или аппаратных средств и программного обеспечения не столь очевидна. Рассмотрим систему, в которой группа процессоров пользуется единственным портом диспетчеризации. Когда один процессор берет некоторый процесс из этого порта (что является неявно заданной операцией), необходимо гарантировать такой режим работы других процессоров, при котором в это время они даже не пытались бы манипулировать портом. Для этой цели первый упомянутый процессор прежде всего проверит, заблокирован ли порт, и, если нет, выполнит его блокирование, воспользуется этим портом и по окончании работы с ним деблокирует его. Если же при обращении к порту процессор обнаружит, что порт заблокирован, процессор будет повторять проверку состояния порта до тех пор, пока последний не окажется свободным.

Необходима также и синхронизация между аппаратно реализованными процессорами и программно реализуемыми процессами. И в этом случае рассмотрим в качестве примера работу порта диспетчеризации, принимая во внимание, что процессоры неявным образом используют информацию о структуре данных, которые могут анализироваться и реорганизовываться программными средствами. Так, если операционной системе необходимо произвести некоторые манипуляции с очередью процессов в порту диспетчеризации, она должна заблокировать порт для гарантии, что процессоры не пользуются им одновременно.

Для блокирования объектов используется последовательность 16 бит, состоящая из 2-битового кода блокировки и 14-битового идентификатора «блокировщика». Двухбитовый код может принимать следующие значения: 00 — объект не заблокирован; 01 — заблокирован (аппаратными средствами путем выполнения неявно заданной операции); 10 — заблокирован (программными средствами с помощью команды LOCK-OBJECT);

11 — блокирован (аппаратными средствами во время выполнения явно заданной операции, например посредством команды SEND, посылающей сообщение в порт). При блокировании аппаратными средствами идентификатор блокирующего процессора помещается в 14-битовое поле идентификатора, при блокировании программными средствами это поле можно использовать для хранения идентификатора блокирующего процесса.

#### НЕДЕЛИМЫЕ ОПЕРАЦИИ ЧТЕНИЯ/ЗАПИСИ

Несколько команд, используемых для извлечения операандов из памяти или записи в нее результата, определены таким образом, что во время выполнения этих команд производится блокировка памяти. К таким командам относится, например, команда INDIVISIBLY-ADD-ORDINAL, при выполнении которой из памяти извлекаются две 32-битовые величины без знака, складываются, а затем записывается результат; при этом гарантируется, что в течение всех этих операций исходные операанды не извлекаются и не модифицируются другим процессором. Данная команда позволяет, например, группе процессов обновлять совместно используемый счетчик.

#### ОБЪЕКТЫ, ТИП КОТОРЫХ ОПРЕДЕЛЯЕТСЯ ПРОГРАММНЫМИ СРЕДСТВАМИ

Рассмотрим еще одну возможность, предоставляемую системой iAPX 432 для защиты объектов от несанкционированного доступа. Система позволяет «запечатать» некоторый произвольный объект таким образом, что им могут манипулировать только те подпрограммы, которым предоставляется право «распечатать» объект (например, подпрограммы домена, представляющего спецификатор типа для данного объекта). Такая возможность реализуется посредством объектов «определение типа» и дескриптора типа в таблице объектов. Каждый объект, защищенный подобными средствами, называется *объектом расширенного типа* или *объектом, тип которого определяется пользователем*.

Эти средства схематически изображены на рис. 17.16. Положим, что имеется объект A, который нужно защитить таким образом, чтобы непосредственно оперировать им имели возможность только те подпрограммы, которые находятся в домене XOPS. Иначе говоря, программам разрешается иметь дескрипторы доступа, которые представляют объект A, но только указанный домен может манипулировать объектом A или наблюдать его состояние. Более того, этому домену разрешается манипулировать только этим объектом или другими идентичными

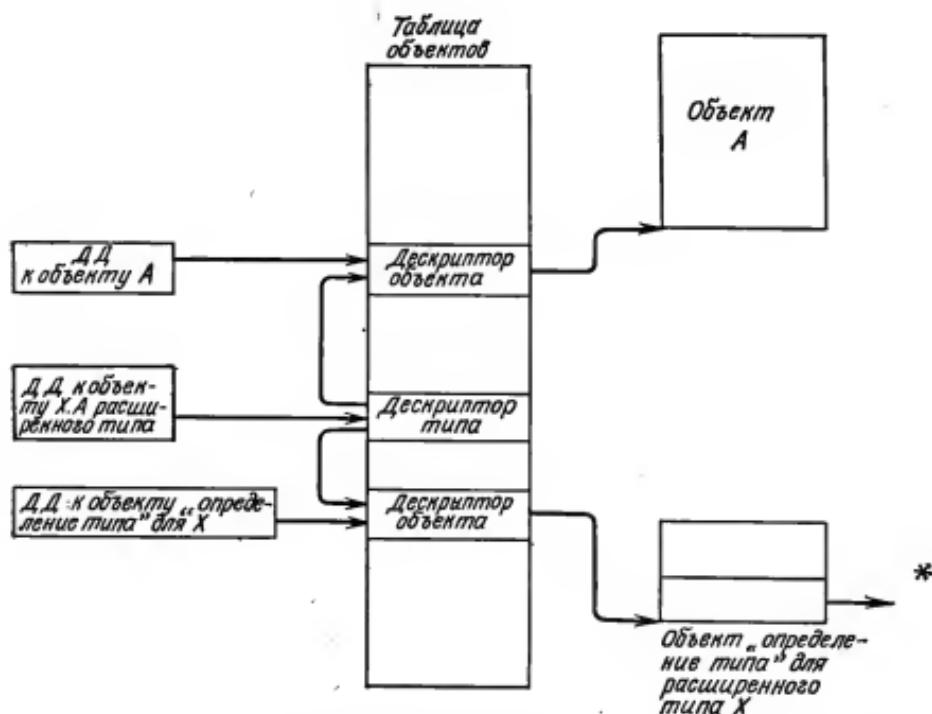


Рис. 17.16. Адресация объекта расширенного типа. (\* — Обычно к домену для спецификатора типа объекта для объектов типа Х.)

структурами (в дальнейшем будем условно называть их структурой типа Х).

Положим, что существует домен XOPS, а также объект «определение типа», представляющий структуру типа Х. Упомянутый объект — это системный объект, состоящий из сегмента доступа. Последний не имеет заранее определенного содержимого, но, как правило, действует соглашение, по которому этот сегмент содержит дескриптор доступа к домену XOPS. Одной из подпрограмм, связанных с этим доменом, может быть подпрограмма CREATE\_X\_OBJECT. Ее функцией является создание объекта типа Х и возврат по адресу вызова дескриптора доступа, представляющего этот объект. После создания объекта А (т. е. объекта типа Х) посредством команды CREATE-DATA-SEGMENT или CREATE-ACCESS-SEGMENT (что делает объект А фактически сегментом доступа или сегментом данных общего назначения) подпрограмма CREATE\_X\_OBJECT выполняет команду CREATE-PRIVATE-TYPE. Двумя операндами этой команды являются дескрипторы доступа для объекта «оп-

		<i>Законсервировано</i>			
<i>ДД для объекта расширенного типа</i>	<i>Номер урбоя</i>		<i>ДД для объекта "определение типа"</i>	<i>Законсервировано</i>	<i>01</i>
<i>Зарезерви- ровано</i>			<i>Любторный запрос памяти</i>	<i>Общее/частное бит готовности</i>	

Рис. 17.17. Представление дескриптора типа в таблице объектов.

ределение типа» и объекта А. Эта команда создает дескриптор типа в таблице объектов, связывая объект А и объект «определение типа», а также создает дескриптор доступа (ДД) к дескриптору типа.

Дескриптор типа, размещающийся в таблице объектов, показан на рис. 17.17. Он содержит дескрипторы доступа для самого объекта, а также для объекта «определение типа». ДД к объекту «определение типа» принято называть дескриптором доступа к объекту расширенного типа. Этого дескриптора недостаточно для выполнения операций над соответствующим объектом, даже если дескриптор и содержит все права доступа; необходимо располагать ДД к самому объекту. Если объект был создан как объект частного пользования (индикатором этого является флаг в дескрипторе типа), ДД к нему может быть получен посредством выполнения команды RETRIEVE-TYPE-REPRESENTATION. Однако указанной команде в качестве операндов требуются ДД к объекту расширенного типа и ДД к объекту «определение типа». Более того, объект «определение типа» должен соответствовать объекту расширенного типа, указанному в дескрипторе типа. Следовательно, если спецификатор типа объекта (ХOPS) содержит ДД к объекту «определение типа», являющийся внутренним по отношению к самому себе (например, в его домене), только ХOPS может непосредственно выполнять операции над объектом расширенного типа.

Мы рассмотрели обычное решение проблемы защиты объектов специального типа от несанкционированного доступа — посредством спецификатора типа объекта. Наряду с этим система предоставляет еще два возможных решения этой проблемы. Первое решение предполагает создание объекта расширенного типа общего пользования (в противоположность объектам частного пользования), что связано с применением команды CREATE-PUBLIC-TYPE. Такой объект имеет также объект «определение типа», а возможно, и спецификатор типа. Однако в этом случае нет ограничений на получение ДД к самому объекту; достаточно располагать ДД к объекту расширенного типа и выполнить команду RETRIEVE-PUBLIC-TYPE-REPRESENTATION. Результат выполнения этой команды подо-

бен результату команды RETRIEVE-TYPE-REPRESENTATION, однако в данном случае не требуется ДД к объекту «определение типа».

Второе решение указанной выше проблемы позволяет путем некоторой процедуры выбора получать ДД к объекту расширенного типа. Известно, что одним из системных прав в ДД к объекту расширенного типа является право брать обратно ДД к объекту «определение типа». Если имеющийся в распоряжении ДД обладает таким правом, можно выполнить команду RETRIEVE-TYPE-DEFINITION, а затем команду RETRIEVE-TYPE-REPRESENTATION.

Объектом расширенного типа можно сделать объект любого типа, включая системные объекты и объекты, которые уже являются объектами расширенного типа.

## УПРАВЛЕНИЕ ПАМЯТЬЮ

В дополнение к возможностям управления объектами и адресации к ним система iAPX 432 предоставляет средства для управления свободной памятью (т. е. памятью, не используемой в текущее время для представления объекта) и выполнения процедуры «сбора мусора» в памяти, занимаемой неиспользуемыми объектами. Многие ранее упоминавшиеся понятия архитектуры системы iAPX 432 имеют отношение к выделению памяти: объект «ресурсы памяти», команды CREATE-ACCESS-SEGMENT и CREATE-DATA-SEGMENT, хип-флаг в дескрипторах доступа, заголовок таблицы объектов, данные объекта «локальные ресурсы памяти» в сегменте данных процесса, команда CALL. С упомянутой процедурой «сбора мусора» связаны следующие понятия: хип-флаг в дескрипторах доступа, номер уровня в дескрипторах объектов, флаг повторного запроса в дескрипторах объектов, команда RETURN.

Память, имеющаяся в распоряжении для выделения объектам, представлена в архитектуре системы iAPX 432 объектами «ресурсы памяти», структура которых показана на рис. 17.18. У этих объектов нет фиксированных взаимосвязей с процессами, процессорами, таблицами объектов и т. п. Количество объектов «ресурсы памяти» в системе и их назначение определяются программными средствами. Сегмент данных такого объекта содержит один или несколько дескрипторов блоков памяти, каждый из которых определяет свободный блок смежных областей памяти путем указания адресов первого и последнего байтов блока (эти байты должны принадлежать 8-байтовым границам памяти). Сегмент данных указывает также индексы первого и текущего (определяемого в дальнейшем) дескрипторов блоков памяти.

Команды CREATE-DATA-SEGMENT и CREATE-ACCESS-SEGMENT присваивают имя объекту «ресурсы памяти» как одному из своих операндов. Машина использует такой объект в качестве источника памяти, необходимой для представления объекта.

Отметим, что объект «процесс» (рис. 17.13) содержит область памяти, помеченную как область объекта «локальные ресурсы памяти» и похожую на сегмент данных объекта «ресурсы памяти» с единственным дескриптором блока памяти. Этот локальный объект «ресурсы памяти» определяет область адресного пространства, которая может быть использована процессом для размещения локальных объектов (например, таким образом машина может выделить место для контекста).

Упомянутые выше команды создания сегментов могут обращаться к объекту «ресурсы памяти» или объекту «локальные ресурсы памяти» (в последнем случае используется селектор операнда 0 в поле операнда команды). В первом случае сегмент

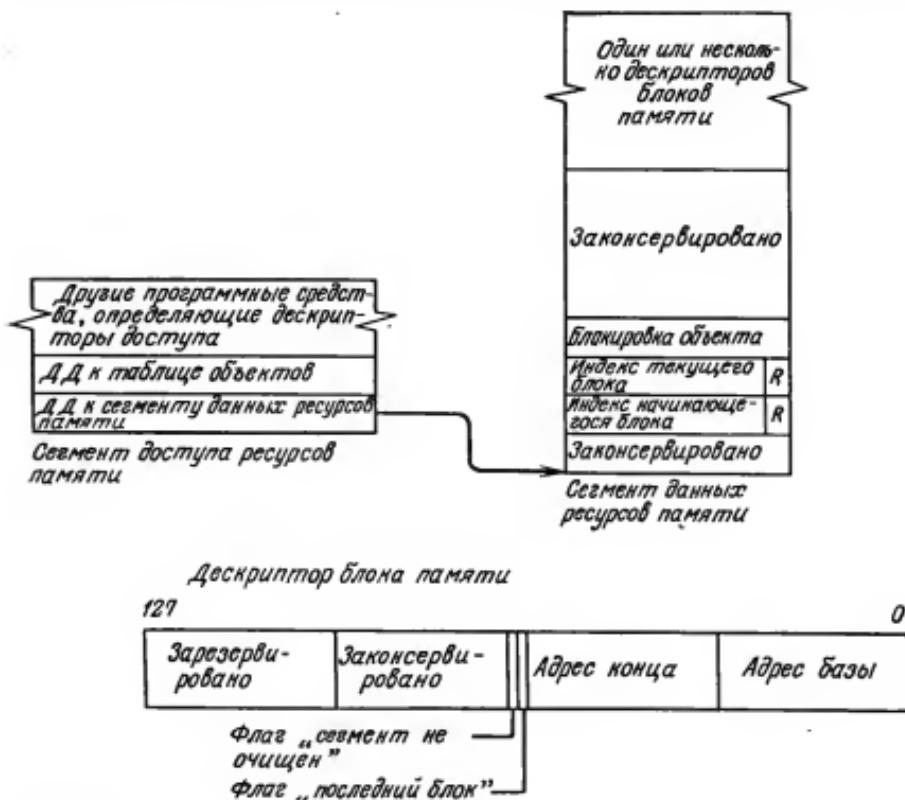


Рис. 17.18. Представление объекта «ресурсы памяти».

считается выделенным из общей, не связанный с процессом памяти (хип-памяти). Это предполагает освобождение этой памяти неявным образом при возвращении из подпрограммы. Хип-флаг в новом дескрипторе доступа устанавливается в 1, тем самым указывая, что общая (глобальная) память была использована для адресуемого сегмента. По тем же причинам в поле номера уровня в новом дескрипторе объекта записывается 0.

Когда процесс создает сегмент, используя данные своего объекта «локальные ресурсы памяти», хип-флаг в новом дескрипторе доступа устанавливается в положение 0, а номер уровня в новом дескрипторе объекта принимает значение, соответствующее текущему контексту в процессе (т. е. значение текущего числа вложенных активаций подпрограммы).

Когда машина адресуется к объекту «ресурсы памяти» для выделения последней, используется специальный алгоритм. (Алгоритм для выделения памяти из объекта «локальные ресурсы памяти» процесса является тривиальным, поскольку в этом случае имеется только один дескриптор блока памяти). Этот алгоритм получил название *циклический поиск первого подходящего случая*; циклический поиск является причиной наличия индекса текущего блока в сегменте данных объекта «ресурсы памяти». Процессор выполняет просмотр дескрипторов блоков памяти в поисках ближайшего, подходящего по объему. Начинается поиск с того дескриптора, который указан индексом текущего блока, и возобновляется, если необходимо, переходом от последнего дескриптора снова к указанному первому. Когда подходящий блок найден, дескриптор блока памяти обновляется, что отображает факт удаления памяти, а индекс текущего блока получает приращение. Если подходящий блок обнаружить не удается, выдается сообщение об ошибке.

После выделения памяти создаются дескриптор объекта (посредством заголовка таблицы объектов, позволяющего обнаружить свободный дескриптор) и дескриптор доступа. Затем осуществляется инициализация сегмента, т. е. присваивается содержимое в виде нулей, если это сегмент данных, или присваивается нулевое значение биту младшего разряда — биту готовности — каждого 32-битового слова, если это сегмент доступа.

Во всех случаях содержимое поля повторного запроса памяти в заголовке таблицы объектов сначала проверяется, а затем уменьшается на величину выделенной памяти. Если результат такого вычитания становится отрицательным числом, выдается сообщение об ошибке.

#### «СБОР МУСОРА» В ПАМЯТИ

Архитектурой системы iAPX 432 предусмотрена возможность выполнения неявным образом задаваемой операции «сбора му-

сора» применительно к локальным объектам. Кроме того, система предоставляет программные средства для подобной операции сбора хип-объектов (т. е. объектов продолжительного времени существования). Когда выполняется команда RETURN, машина повторно запрашивает всю память, связанную с объектами, номера уровней которых больше или равны номеру уровня текущего контекста. Это осуществляется путем поиска местоположения объектов посредством текущей таблицы объектов данного процесса.

Казалось бы, это должно создавать проблему не имеющих адресата ссылок, например в случае, когда контекст возвращается дескриптор доступа к локальному объекту в качестве параметра или когда он записывает дескриптор доступа к локальному объекту в хип-объект. Однако архитектура системы iAPX 432 исключает возникновение подобных проблем. Если хип-флаг дескриптора доступа не установлен в 1, этот дескриптор не может быть записан в сегмент, номер уровня которого меньше номера уровня текущего контекста.

Что же касается хип-объектов, то, согласно их определению как объектов продолжительного времени существования, проблема ссылок, не имеющих адресата, для них не возникает. Отметим, что принципы архитектуры системы iAPX 432 базируются на существовании программно реализуемого алгоритма поиска и «сбора» областей памяти, занимаемых объектами, возможность адресации к которым утеряна. Этот алгоритм представляет собой описание процесса, состоящего из двух фаз и протекающего одновременно с другими процессами системы; его функции — идентификация и сбор сегментов, не имеющих ссылок [1]. Алгоритм предъявляет к системе адресации только одно требование — иметь возможность помечать (выставлять «серый флаг») сегмент, адрес которого зарегистрирован. Для этой цели используется флаг повторного запроса памяти в дескрипторах в таблице объектов. Соответствующему биту процессор присваивает значение 1 при создании или копировании дескриптора доступа, связанного с тем или иным элементом таблицы объектов.

## ОБРАБОТКА ОШИБОК

В архитектуре системы iAPX 432 предусмотрены средства обработки ошибок (обнаруживаемых машиной), которые возникают при работе процессора, а также выполнении операций процесса и контекста. При этом система выдает достаточно полную информацию об ошибке, позволяющую программе обработки ошибок выполнить коррекцию и продолжить выполнение программы.

В соответствии с «многоуровневым» описанием функционирования системы (уровень контекста, процесса, процессора) ошибки подразделяют на ошибки на уровне процессора, процесса и контекста. Ошибки на уровне контекста порождаются программными ошибками (например, ошибки выполнения вычислений, индексирования, неполного задания объекту необходимых прав). Каждый объект «команды» в соответствующем ему объекте «домен» содержит индекс дескриптора доступа другого объекта «команды» (программы обработки ошибок), которому следует передать управление при возникновении ошибок на уровне контекста. Войти в программу обработки ошибок можно посредством неявно заданной команды перехода; при этом команды программы обработки ошибок выполняются как часть текущего контекста. В сегменте данных контекста имеется область хранения информации об ошибке, куда и помещаются сведения об обнаруженной ошибке. После выполнения этих действий ответственность за обработку ошибки на уровне контекста возлагается на объект «команды обработки ошибок». Если упомянутый выше объект «команды» не указывает на подобную программу обработки ошибок, то ошибка на уровне контекста становится ошибкой на уровне процесса.

Отличительными признаками ошибок на уровне процесса являются следующие: 1) взаимосвязь с «окружением» процесса в системе во время его протекания (например, с информацией о блокировании объектов, об управлении памятью, о присутствии в системных объектах информации, не соответствующей данным условиям протекания процесса) или 2) принадлежность к ошибкам на уровне контекста, для которых объект «команды» не указывает программу обработки ошибок. Специфичным является и способ обработки ошибок на уровне процесса. Каждый сегмент доступа процесса (рис. 17.13) может содержать дескриптор доступа, адресующийся к порту ошибок. При возникновении ошибки на уровне процесса в область информации об ошибке в объекте «процесс» записывается некоторое значение, и процессор выполняет неявно заданную операцию пересылки объекта «процесс» как сообщения в порт ошибок. Затем процессор опрашивает свой порт диспетчеризации в поисках другого процесса. В операционной системе будет находиться процесс — программа обработки ошибок, принимающий содержащие ошибки процессы из упомянутого порта ошибок (представляющего собой нормальный порт связи).

В поле состояния сегмента данных процесса имеется флаг, указывающий на необходимость обработки ошибок на уровне процесса как ошибок на уровне контекста.

К ошибкам на уровне процессора относятся, во-первых, ошибки, вызываемые машинными (аппаратными) сбоями; во-

вторых, ошибки, возникающие, когда у процессора нет назначенного ему процесса (например, ошибки, связанные с функционированием порта диспетчеризации); в-третьих, ошибки на уровне процесса, которые не поддаются обработке (например, если либо не указан порт ошибок для процесса, либо возникает ошибка при выполнении неявно задаваемой операции пересылки содержащего ошибку процесса, либо оказывается заполненной очередь в порту ошибок).

При возникновении ошибок на уровне процессора соответствующая информация помещается в область информации об ошибке в сегменте данных процессора. Процессор отыскивает свой порт диагностики, указываемый в объекте «процессор», и посыпает процесс диагностики в очередь этого порта. Если процессор располагает назначенным ему процессом, последний не помещается в очередь порта диспетчеризации.

Нельзя, конечно, исключать возможность неудачного выполнения описанных выше процедур (например, вследствие отказов в работе электронных схем процессора, отсутствия порта диагностики, сбоев в процессе адресации процессора к порту диагностики). В таком случае имеет место прекращение работы процессора — его останов, что сопровождается формированием соответствующего сигнала на специальном выводе.

#### типы ошибок

Ниже перечисляются некоторые виды ошибок и уровни, на которых они имеют место.

Уровень	Название ошибки
Контекст	Переполнение индекса при масштабировании > Переполнение смещения при адресации > Переполнение указателя команды > Наличие ошибки в домене (недопустимый класс, деление на нуль) > Переполнение > Антипереполнение > Недостаточная точность > Недопустимый системный тип (несоответствие типа команды/объекта) > Переполнение сегмента > Переполнение памяти > Нарушение прав на чтение > Нарушение прав на запись > Некорректное использование прав доступа к адресату сегмента доступа
Процесс	Неудачная попытка обработки ошибок на уровне контекста > Наличие ошибки в дескрипторе объекта > Недействительность дескриптора доступа (неопределенный ДД)

Уровень	Название ошибки
Процесс	Преждевременное использование права на удаление адресата Переполнение индекса объекта «команды» Антипереполнение запроса памяти в заголовке таблицы объектов Стирание ДД (путем записи другой информации), флаги удаления которых не сброшены Стирание уровня дескриптора доступа Выход за пределы допустимых значений индекса входного элемента Ошибочное указание прав/типа объекта «управление дескриптором»/«управление аффинажем»/«описание типа» Нарушение системных прав Неправильное задание размеров параметров контекста Преждевременное изменение дескриптора доступа. (В процессе выполнения команды AMPLIFY-RIGHTS дескриптор доступа изменяется прежде, чем результат выполнения команды записывается в память.) Нарушение требований блокировки объектов (попытка выполнять операции над системными объектами без блокирования этих объектов) Переполнение аффинажа Состояние, при котором исчерпаны все ресурсы дескрипторов объектов Наличие неправильного адреса в дескрипторе блока памяти Переполнение индекса дескриптора блока памяти Фрагментация блока памяти Некорректное извлечение команды Неудачная попытка обработки ошибки на уровне процесса Регистрация ошибки при отсутствии процесса Сбои аппаратных средств
Процессор	

#### ОБЛАСТЬ ИНФОРМАЦИИ ОБ ОШИБКЕ

В сегментах данных объектов «процессор» и «процесс» имеется область, используемая машиной для описания ошибки. На рис. 17.19 показан формат такой области.

В поле индекса объекта с ошибкой размещается ссылка на дескриптор доступа объекта «команды», содержащего ошибку и расположенного в домене, который связан с данным объектом с ошибкой. Поля указателей стеков предшествующей и последующей команд содержат смещения (в битах) команды, при выполнении которой регистрируется ошибка, и следующей за ней команды. (Если ошибка возникает до завершения кодирования команды, содержимое поля указателя последующей команды может оказаться непредсказуемым.) Аналогичным образом поля указателей стеков предшествующей и последующей команд содержат значения указателя стека операндов соответственно во время начала выполнения команды и при возникновении ошибки.

В поле ошибки имеются четыре флага, описывающие характеристики команды, при выполнении которой возникает ошибка, и состояние контекста. Эта информация предназначена для программы обработки ошибок. Упомянутые флаги указывают следующее: является ли операнд, адресующий результат выполнения команды, ссылкой на стек или ссылкой на область памяти; допускается ли получение неточных результатов; заполнен ли регистр содержимого вершины стека к началу выполнения команды или к моменту возникновения ошибки. Если флаг 1 указывает, что результат подлежит пересылке в память, поля селектора сегмента с ошибкой и смещения ошибки содержат информацию соответственно о селекторе сегмента, в котором обнаружена ошибка, и о смещении в сегменте доступа (или сегменте данных, к которым уже получен доступ) в момент возникновения ошибки.

Поля первого операнда — источника и второго операнда — источника (или результата, соответствующего так называемой исключительной ситуации) используются для команд выполнения вычислительных операций. Если ошибка происходит до выполнения командой соответствующей операции, то указанные поля содержат значения операндов — источников. Если же ошибка возникает после формирования результата (например, ошибка переполнения), второе из названных выше полей содержит результат, представленный в так называемой исключительной форме.

Поле индекса кода операции может быть использовано вместе с соответствующей таблицей кодов операций для определения кода операции команды, вызвавшей появление ошибки.

Поле кода ошибки содержит 16 бит информации о типе ошибки. Кодирование этой информации осуществляется по весьма сложному правилу и здесь рассматриваться не будет. В некоторых случаях информация этого поля подлежит использованию вместе с информацией о типе команды для определения типа ошибки. Для ошибок, связанных с адресацией к памяти, это поле содержит индикатор вида операции («чтение» или «за-

Первый операнд-источник
Второй операнд-источник или исключительный результат
Смещение ошибки
Селектор сегмента с ошибкой
Код ошибки
Индекс кода ошибки.
Поле ошибки
Указатель стека предшествующей команды
Указатель стека последующей команды
Указатель предшествующей команды
Указатель последующей команды
Индекс объекта с ошибкой

Область информации об ошибке

Рис. 17.19. Формат области, содержащей информацию об ошибке в объектах «процесс» и «процессор».

пись»). Для ошибок, выражающихся в переполнении сегмента, нарушении прав на чтение или запись или несоответствии типов команд и объекта, в поле кода ошибки содержится информация о типе объекта, к которому осуществляется доступ.

## ОСНОВНЫЕ ХАРАКТЕРИСТИКИ

### СИСТЕМЫ iAPX 432

Сравним системы iAPX 432 и SWARD. Их сопоставление интересно уже потому, что при разработке этих систем были поставлены одни и те же цели (облегчение процесса разработки программного обеспечения), а многие основные принципы их построения на самом высоком уровне абстракции сходны (например, в обеих системах используются такие абстрактные понятия, как объекты, потенциальные адреса, одновременно протекающие процессы). Главное различие между этими системами заключается в том, что упомянутые принципы построения системы SWARD представляют собой абстракции, создаваемые при реализации машины (речь идет об объектах, таких, как процесс-машины), в то время как подобным абстракциям в системе iAPX 432 соответствуют доступные пользователю структуры данных. Однако последние с помощью средств операционной системы могут быть «скрыты» от пользователя и стать тем самым для них недоступными. На основании подобного сравнения двух указанных систем можно сделать следующие (хотя и не проверенные на практике, а поэтому умозрительные) заключения:

- 1) архитектура системы SWARD менее сложная, чем архитектура системы iAPX 432;
- 2) аппаратные средства системы iAPX 432 по своей организации проще, чем средства системы SWARD;
- 3) система SWARD отличается более высоким быстродействием, поскольку предоставляет больше возможностей по распараллеливанию функций, подлежащих выполнению;
- 4) системе iAPX 432 присуща большая гибкость, перестраиваемость конфигурации средств;
- 5) система iAPX 432 гарантирует меньший риск отказов по сравнению с системой SWARD.

Применяемое к системе iAPX 432 утверждение, что это — машина операционной системы, наиболее выразительно характеризует специфику данной вычислительной системы. (Иногда ее называют машиной высокого уровня языка Ада; однако такое определение нельзя признать корректным, поскольку система iAPX 432 обеспечена ограниченными средствами, имеющими прямое отношение к реализации языка Ада.) Система iAPX 432 располагает обширным набором средств управления памятью, планирования процессов и организации коммуникаций между

процессами, позволяющим сравнительно легко реализовать ту или иную стратегию функционирования операционной системы. Ниже перечисляются основные специфические характеристики системы iAPX 432.

*Описание системы посредством объектов.* Такой прием позволяет добиться высокой однородности, регулярности средств описания. Например, все компоненты системы (даже такие, как процесс или процессор) могут быть представлены объектами «сообщение», пересылаемыми через порт. Используя объекты как единую форму описания архитектуры системы, удается с общих позиций определить взаимосвязь машины, операционной системы, языка программирования и прикладных программ.

*Использование средств защиты от несанкционированного доступа.* Система располагает широким набором средств защиты наряду с механизмом потенциальной адресации. Этот набор включает права системы, объекты «управление дескриптором» и «определение типа». Наиболее мощным средством являются объекты «управление дескриптором», поскольку они представляют широкий выбор вариантов реализации своих возможностей. Например, объект «управление дескриптором» может быть определен для создания объектов «порт», расширения любых прав доступа, предоставления только права на чтение или использования объектов «процесс».

*Ориентация на работу с программами высокой модульности при использовании спецификаторов типов объектов.* Многое из того, что положено в основу системы, связано с указанной ориентацией: объекты «домены», «контексты», «определение типа», «аффинаж» и средства расширения прав доступа.

*Наличие параллельных процессов и группы процессоров.* Имеются чрезвычайно гибкие разнообразные программные средства организации их совместной работы. Соответствующие средства, обеспечивающие их функционирование, включают возможность задания, во-первых, правил организации очередей, специфических для каждого порта; во-вторых, параметров приоритета и гарантированных средств обслуживания каждого процесса; в-третьих, временного интервала и пределов периода обслуживания каждого процесса; в-четвертых, порта планирования.

*Возможность использования одного и того же набора средств блокирования объектов как аппаратными, так и программными средствами.* Эти средства, использование которых в системе iAPX 432 необходимо, являются нововведением в архитектуре ЭВМ.

*Однородность архитектуры системы.* Предполагается применимость принципов построения архитектуры системы iAPX 432 при системном проектировании разнородных процессоров. (В су-

ществующих в настоящее время проектах имеются процессоры двух видов — обработки данных общего назначения и интерфейсные.)

Как и в случае системы SWARD, использование языка Ада в данной системе наталкивается на определенные несоответствия между требованиями языка и возможностями системы. Причиной этого является привязка программ на языке Ада к окружающим их средствам системы уже на ранней стадии обслуживания задания, что в свою очередь объясняется ориентацией разработчиков этого языка на традиционные вычислительные системы. Например, принцип использования домена (для пакетов языка Ада) для описания операций на этапе выполнения, потенциальные адреса, объекты расширенного типа, объекты «управление дескриптором» и другие компоненты архитектуры системы iAPX 432 оказываются иенужными как средства, окружающие программу на языке Ада, хотя перечисленные компоненты дают возможность создания более динамичного и надежного окружения для программ, чем то, которое запрашивается правилами этого языка. Сотрудники фирмы Intel для решения этой проблемы предложили расширить некоторые средства языка Ада с целью реализации последнего в системе iAPX 432 с учетом широких возможностей этой вычислительной системы.

Сравнивая относительное быстродействие рассматриваемых вычислительных систем, следует отметить, что в существующем прототипе системы SWARD, несмотря на более низкую частоту ее тактового генератора и высокую частоту проверки содержимого регистров, программы выполняются несколько быстрее, чем в процессоре обработки данных общего назначения (GDP) системы iAPX 432. (Однако при рассмотрении отдельных случаев такой вывод может оказаться неверным; так, в системе iAPX 432 быстрее выполняются операции арифметики чисел с плавающей точкой, а система SWARD обеспечивает более высокую скорость выполнения операций над строками символов.) Все перечисленное объясняется различиями в архитектурах сравниваемых систем, а также спецификой конкретной реализации каждой из них. Что касается архитектуры, то в системе iAPX 432 объекты (в частности, объект «контекст») представлены как группы сегментов памяти, что влечет необходимость повторного выделения памяти при обращении к подпрограммам. И если продолжить сравнение способов обращения к подпрограммам в обеих системах, то следует указать, что действия, реализуемые командой CALL в системе SWARD, выполняются в системе iAPX 432 с помощью последовательности команд, которые должны генерировать компилятор (команды для создания списка формальных параметров в сообщении, команды присвоения

начальных значений локальным переменным и формирования сегментов доступа к точкам входа в подпрограмму).

Продолжив сравнение систем iAPX 432 и SWARD, следует отметить, что первая из них при каждом обращении к данным должна решать проблему определения одного или нескольких потенциальных адресов путем прохода нескольких уровней косвенной адресации, в то время как вторая система не нуждается в средствах потенциальной адресации для доступа к локальной переменной или к параметру. Для компенсации этих недостатков адресации в системе iAPX 432 предусмотрены два ассоциативных запоминающих устройства преобразования адресных ссылок (см. гл. 18).

Специфика конкретной реализации той или иной системы затрудняет сравнительную оценку их эффективности. Отметим только, что в существующем в настоящее время процессоре обработки данных общего назначения системы iAPX 432 используется микропрограмма последовательной организаций алгоритма функционирования, тогда как у созданного прототипа системы SWARD микропрограмма отличается сравнительно высокой степенью параллелизма выполнения требуемых функций.

## УПРАЖНЕНИЯ

17.1. Объясните, с какой целью в дескрипторе доступа осуществляется разграничение прав объекта и системных прав.

17.2. Укажите базовый тип сегментов объектов «таблица объектов» и «справочник таблиц объектов».

17.3. Поясните, каким образом операционная система осуществляет пуск процесса после того, как к нему созданы сегменты, образующие объект «процесс».

17.4. Дайте ответ, почему по команде SEND осуществляется блокирование процесса, транспортер которого использован для постановки в очередь сообщения.

17.5. Спецификатор типа объектов располагает средствами защиты объектов от несанкционированного доступа. Сопоставьте функции используемых для этих целей объектов «управление дескриптором» совместно со средствами расширения прав доступа, с одной стороны, и объектов «определение типа», с другой.

17.6. При выполнении команд CREATE-DATA-SEGMENT и CREATE-ACCESS-SEGMENT в таблице объектов создается дескриптор объекта. Укажите, в какой именно таблице из всех возможных таблиц объектов системы.

17.7. Объясните, с какой целью при создании сегмента доступа машина присваивает значение 1 младшему биту 32-битового слова.

17.8. Укажите, в каких случаях возникает ситуация, при которой программе желательно указывать в соответствующем объекте «процесс», что ошибки на уровне процесса следует воспринимать как ошибки на уровне контекста.

17.9. Поясните, в каких ситуациях сегмент команд, обрабатывающих ошибки, может использовать команду SET-CONTEXT-MODE.

## ЛИТЕРАТУРА

1. Dijkstra E. W., On-the-Fly Garbage Collection. An Exercise in Cooperation, *Communications of the ACM*, 21(11), pp. 966—975 (1978).

---

## ГЛАВА 18

### НАБОР КОМАНД ПРОЦЕССОРА ОБЩЕГО НАЗНАЧЕНИЯ iAPX 432

В данной главе рассматриваются форматы и назначение команд процессора общего назначения системы iAPX 432. В последнем разделе описывается адресная информация, хранимая в блоках ассоциативной памяти, используемых для преобразования адресов.

#### ФОРМАТЫ КОМАНД

Все машинные команды системы iAPX 432 имеют один и тот же формат (рис. 18.1). Каждая команда представляется четырьмя полями. Первым является *поле класса команды*. В нем определяется число операндов и их размеры. Во втором поле — так называемом *поле формата* — задается порядок размещения операндов и указывается, представлены они адресами данных в памяти или находятся в стеке. Третье поле — *поле адресов данных* или *адресов перехода*. Четвертое поле содержит *код выполняемой операции (КОП)*. Содержимое поля класса и содержимое поля КОП не являются независимыми; в вычислительных системах другой архитектуры содержимое этих полей обычно объединено и занимает поле КОП. Наибольшая допустимая длина команды равна 321 бит, наименьшая — 6 бит.

Принятый формат команд призван оптимизировать кодирование данных и интерпретацию команды при ее выполнении. Как отмечалось выше, команды могут размещаться в сегменте команд со смещением на произвольное число битов. Процессор анализирует содержимое поля класса команды и определяет число операндов и их размеры. После того как посредством такого анализа будут извлечены операнды, выполняется анализ КОП и выясняется, что именно должно быть сделано с операндами: вычислена сумма, произведение и т. п. В табл. 18.1 приведено возможное содержимое поля класса команды.

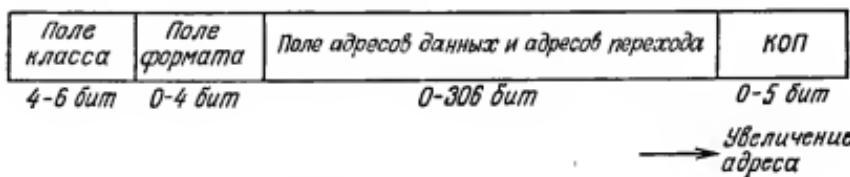


Рис. 18.1. Формат команд системы iAPX 432.

Содержимое поля формата определяет следующее: 1) представляет ли операнд обращение к данным из основной памяти или является указателем на необходимость извлечения операнда из стека (в последнем случае соответствующий операнд команды не является адресом данных); 2) какова позиция в команде обращения к данным, которая соответствует определенному операнду (например, первому операнду команды ADD-INTEGER не обязательно должен соответствовать первый адрес данных в этой команде). Возможное содержимое поля формата приведено в табл. 18.2. Содержимое поля формата взаимосвязано с содержимым поля класса команды. Так, длина поля формата зависит от числа operandов, производящих обращение к данным (в отличие от адресов перехода). Это число определяется в поле класса команды. Если в команде нет operandов, адресу-

Таблица 18.1. Содержимое поля класса команды

Содержимое поля класса	Размеры operandов	Содержимое поля класса	Размеры operandов
001110	Отсутствуют	101101	64 80
101110	Адрес перехода	011101	80 8
0000	8 адрес перехода	111101	80 32
011110	8	000011	80 64
111110	16	100011	80 80
000001	32	010011	8 8 8
100001	64	1100	16 16 8
010001	80	0010	16 16 16
110001	8 8	110011	16 16 32
001001	8 16	001011	16 32 32
101001	16 8	1010	32 32 8
1000	16 16	0110	32 32 32
011001	16 32	101011	32 32 80
111001	16 80	011011	32 80 80
000101	32 8	111011	64 64 8
100101	32 16	000111	64 64 80
0100	32 32	100111	64 80 80
010101	32 80	010111	80 32 80
110101	64 8	110111	80 64 80
001101	64 64	001111	80 80 8
		101111	80 80 80

Таблица 18.2. Содержимое поля формата команды

Количество операндов	Содержимое поля формата	Значения, соответствующие		
		операнду 1	операнду 2	операнду 3
0	Отсутствует			
1	0	АД1		
1	1	Стек		
2	00	АД1	АД2	
2	10	»	АД1	
2	01	»	Стек	
2	011	Стек	АД1	
2	111	»	Стек	
3	0000	АД1	АД2	АД3
3	1000	»	»	АД2
3	0100	»	»	АД1
3	1100	»	»	Стек
3	0010	»	Стек	АД2
3	1110	Стек	АД1	»
3	1010	АД1	Стек	АД1
3	0001	Стек	АД1	»
3	0110	АД1	Стек	Стек
3	1001	Стек	АД1	»
3	0111	Стек 1	Стек 2	АД1
3	0101	Стек 2	Стек 1	»
3	1011	Стек 1	Стек 2	Стек
3	1101	Стек 2	Стек 1	»
3	0011	АД2	АД1	АД3
3	1111	»	АД1	Стек

Обозначения. АД — адрес данных в оперативной памяти, «Стек» и «Стек 1» — данные на вершине стека, «Стек 2» — данные, следующие за данными «Стек 1».

ющихся к данным (содержимое поля класса команды равно 001110 или 101110), то поле формата отсутствует.

Из табл. 18.2 следует, что содержимое поля формата может служить указателем и тех ситуаций, когда число обращений к данным меньше числа операндов команды, даже если стек не используется. Например, для команды с числом операндов, равным 2, и содержимым поля формата, равным 10, оба операнда задаются содержимым одного и того же (единственного) поля адреса данных. В соответствии с этим для выполнения оператора присваивания  $A:=A+1$  с целочисленным значением A можно воспользоваться командой INCREMENT-INTEGER (с двумя operandами для исходного значения и для результата), задав в команде только один адрес данных (A). Таким образом, в системе iAPX 432 проводится различие между числом

операндов команды и количеством адресов операндов, задаваемых в команде.

В следующем поле команды содержатся обращения к данным (адреса для всех операндов, не находящихся в стеке операндов) и адрес перехода (адрес команды). Общие принципы адресации operandов рассмотрены в гл. 17, поэтому здесь будет описан только способ их кодирования. При необходимости читатель может обратиться к рис. 17.7—17.9, на которых в форме графов древовидной структуры систематизированы все возможные варианты адресов данных.

На рис. 18.2 в так называемой нормальной бэкусовской форме приведены правила кодирования поля адресов данных и адреса перехода. Чтобы упростить подобную форму записи, из описания исключены взаимозависимости между отдельными элементами (например, в терме «скаляр» терм «смещение» должен иметь вид «смещение 16», если терм «тип длины поля смещения» равен 1).

Следует обратить внимание, что в описании полей команды, приведенном на рис. 18.2, порядок их перечисления является противоположным изображенному на рис. 18.1. На рис. 18.1 поля в сегменте команды расположены по возрастанию адресов. Так, поле класса в команде имеет меньшее значение адреса, чем поле КОП. Однако с целью достижения соответствия с требованиями спецификаций фирмы Intel для системы iAPX 432 и общепринятым соглашением о расположении старших значащих битов в левой части поля представления в тексте на рис. 18.2 для команд и их отдельных полей биты младших разрядов адреса размещены справа. При этом можно сказать, что процессор «просматривает» поля команд, начиная слева, т. е. с поля класса.

В командах передачи управления задается адрес перехода, структура которого также отражена в бэкусовской записи (рис. 18.2). Поле адреса перехода содержит бит, значение которого определяет, какая из двух форм представления адреса используется, а также 10-битовое смещение со знаком или 16-битовое смещение без знака, указывающее команду, которой должно быть передано управление. Смещение со знаком, или относительное смещение, определяет положение адресуемой команды относительно адреса данной команды. Смещение без знака, или абсолютное смещение, задает положение адресуемой команды относительно начала сегмента команд. Значения смещений выражаются в битах, поскольку команды могут начинаться с любого бита. Это объясняет, почему максимальная длина сегментов команд составляет 8192 байт.

Для многих команд один или несколько operandов являются объектами или дескрипторами доступа. Адресация к ним вы-

команда ::= КОП [адреса_операндов] [формат] класс	
адреса_операндов	::= обращение_к_данным
	обращение_к_данным
	обращение_к_данным обращение_к_данным
	обращение_к_данным обращение_к_данным
	обращение_к_данным обращение_к_данным
	адрес_перехода
	обращение_к_данным адрес_перехода
обращение_к_данным	::= адрес_скаляра
	адрес_элемента_записи
	адрес_элемента_вектора
	адрес_элемента_вектора_динамического_типа
адрес_скаляра	::= смещение_селектор_сегмента
	тип_длины_поля_смещения
	тип_селектора_сегмента 00
адрес_элемента_записи	::= косвенное_представление_базы смещение
	селектор_сегмента тип_длины_поля_смещения
	тип_селектора_сегмента 01
адрес_элемента_вектора	::= косвенное_представление_индекса база
	селектор_сегмента тип_длины_базы
	тип_селектора_сегмента 10
адрес_элемента_вектора_динамического_типа	::= косвенное_представление_индекса
	косвенное_представление_базы селектор_сегмента
	тип_селектора_сегмента 11
тип_селектора_сегмента	::= 00 короткий_прямой
	01 длинный_прямой
	10 косвенный_посредством_стека
	11 косвенный_общего_назначения
селектор_сегмента	::= селектор_короткий_прямой
	селектор_длинный_прямой
	селектор_косвенный_посредством_стека
	селектор_косвенный_общего_назначения
селектор_короткий_прямой	::= индекс_BCD_4 ВСД_2
селектор_длинный_прямой	::= индекс_BCD_14 ВСД_2
селектор_косвенный_посредством_стека	::= не кодируется
селектор_косвенный_общего_	

назначения	::= смещение_7 ВСД_2 индекс_ВСД_2 00 ВСД_2  смещенис_7 ВСД_2 индекс_ВСД_6 01 ВСД_2  смещенис_16 ВСД_2 индекс_ВСД_2 10 ВСД_2  смещение_16 ВСД_2 индекс_ВСД_6 11 ВСД_2
косвенный_индекс	::= косвенное_представление_базы/индекса
косвенная_база	::= косвенное_представление_базы/индекса
косвенное_представление_базы/индекса	::= 1 обращение к стеку базы/индекса
типы_селектора_сегмента_длинного/короткого	смещение тип_длины_поля_смещения 10 внутрисегментное обращение  смещение селектор_сегмента_длинный/короткий длина_смещения тип_селектора_сегмента_длинного/короткого 00 короткое обращение к базе/индексу общего назначения
типы_селектора_сегмента_длинного/короткого	::= 0 короткий_прямой_селектор  1 длинный_прямой_селектор
селектор_сегмента_длинный/короткий	::= короткий_прямой_селектор  длинный_прямой_селектор
типы_длины_поля_смещения	::= 0 7-битовое смещение 1 16-битовое смещение
смещение	::= смещение_7  смещение_16
типы_длины_базы	::= 0 нулевая длина (отсутствие) базы  1 16-битовое смещение базы
база	::= [смещенис_16]
адрес_перехода	::= относительный_10 0  абсолютный_16 1
смещенис_7	7-битовое смещение без знака (в байтах)
смещение_16	16-битовое смещение без знака (в байтах)
ВСД_2	2 бит селектора ВСД
индекс_ВСД_2	2 бит индексирования ДД в ВСД
индекс_ВСД_4	4 бит индексирования ДД в ВСД
индекс_ВСД_14	14 бит индексирования ДД в ВСД
относительный_10	-10-битовое смещение со знаком (в битах)
абсолютный_16	16-битовое смещение без знака (в битах)

Рис. 18.2. Синтаксис команды системы iAPX 432.

полняется следующим образом. С помощью любой формы обращения к данным определяется адрес короткого порядкового числа как искомого операнда. Для извлечения этого числа (адресации к нему) используется заданное смещение в соответствующем сегменте. Полученное число воспринимается как значение длинного прямого селектора сегмента, т. е. как 14-битовый индекс дескриптора доступа (ДД) в ВСД и 2-битовый селектор ВСД. Если в соответствии с типом команды операнд является дескриптором доступа, этим операндом и будет ДД, определенный этим селектором в виде короткого порядкового числа. Если же операндом является объект, он определяется указанным ДД.

Специфика использования рассматриваемого формата команд заключается в том, что в соответствии со структурой этого формата максимальное количество операндов равно трем. В то же время для некоторых команд, выполняющих операции над объектами (это будет видно из следующего ниже описания), требуется большее число операндов. Например, операндами команды CREATE-GENERIC-REFINEMENT являются объект «ресурсы памяти», два ДД и два коротких порядковых числа. Увеличить фактическое число операндов удается следующим образом. Согласно формату данной команды, для трех обращений к данным используется одно 16- и два 32-битовых поля. Шестнадцатибитовое поле предназначено для длинного прямого селектора сегмента объекта «ресурсы памяти». Первое из 32-битовых полей содержит два длинных прямых селектора сегментов для двух дескрипторов доступа, а во втором 32-битовом поле находятся два операнда — адреса двух коротких порядковых чисел. В нижеследующем описании команд такие подробности опускаются.

Поле кода операции совместно с полем класса команды определяет тип выполняемой операции. В некоторых командах поле КОП отсутствует, поскольку в них тип операции однозначно определяется содержимым поля класса (например, класс 001110 определяет команду возврата, а класс 101110 — команду безусловной передачи управления). Длина поля КОП (если оно используется) может быть величиной от 1 до 5 бит.

Рассмотрим пример кодирования команды. Пусть в 32-битовом поле со смещением 1E в сегменте данных контекста находится значение переменной I, а в 32-битовом поле со смещением 79 — значение переменной J. Из структуры объекта «контекст» (рис. 17.11) следует, что переменные I и J будут адресоваться посредством ВСД со значением 0 (BCD 0 — сегмент доступа к контексту) и индекса ВСД, равного 0 (ДД к сегменту данных контекста располагается в контексте со смещением 0). Рассмотрим команду ADD-INTEGER со значениями КОП и

класса соответственно 0101 и 0110. (Команде этого класса, как следует из табл. 18.1, соответствует команда с тремя 32-битовыми операндами.)

Для выполнения операции  $I := I + J$  команда должна быть закодирована в виде:

0101	1111001	0000 00 0 00 00	0011110	0000 00 0 00 00	0100	0110
КОП	Обращение к данным 2		Обращение к данным 1		Формат	Класс

Согласно содержимому поля формата, поле обращения к данным 1 соответствует первому и третьему операндам, а поле обращения к данным 2 — второму операнду. В поле обращения к данным 1 справа налево закодирована следующая информация: адрес скаляра (00), короткий прямой селектор сегмента (00), тип 0 длины поля смещения, соответствующий 7-битовому смещению, ВСД 0 (00), индекс ВСД — 0000 и величина смещения 1E.

Для суммирования значений двух верхних слов стека и загрузки результата в поле переменной I следует воспользоваться командой

0101	0011110	0000 00 0 00 00	0111	0110
КОП	Обращение к данным 1		Формат	Класс

## КОМАНДЫ ОБЩЕГО НАЗНАЧЕНИЯ

В этом и следующих разделах рассматриваются команды процессора общего назначения системы iAPX 432. Порядок рассмотрения в основном тот же, что и в гл. 15. Поскольку значительное количество команд имеет много общего, при дальнейшем рассмотрении они объединены в группы. После имени команды приводятся значения ее КОП и класса, например 00..110001 или ..001001, если поле КОП отсутствует.

*Имя команды.* MOVE-CHARACTER 00..110001

*Имя команды.* MOVE-SHORT-ORDINAL 0000..1000

*Имя команды.* MOVE-ORDINAL 000..0100

*Имя команды.* MOVE-REAL 0..001101

*Имя команды.* MOVE-TEMPORARY-REAL 00..100011,

*Выполняемая операция.* Содержимое 8-, 16-, 32-, 64- или 80-битового поля операнда — исходных данных копируется в поле операнда — результата.

*Количество operandov.* 2

*Примечание.* Операции над стеком — загрузка и извлечение данных — выполняются, если в поле формата в качестве одного из operandов указан стек.

**Имя команды.** SAVE-CHARACTER 11..011110

**Имя команды.** SAVE-SHORT-ORDINAL 010..11111

**Имя команды.** SAVE-ORDINAL 010..000001

**Имя команды.** SAVE-REAL 1..100001

**Имя команды.** SAVE-TEMPORARY-REAL 1..010001

**Выполняемая операция.** 8-, 16-, 32-, 64- или 80-битовое содержимое вершины стека копируется в поле, адресуемое операндом команды. Состояние стека не изменяется.

**Количество operandов.** 1

**Имя команды.** ZERO-CHARACTER 0..011110

**Имя команды.** ZERO-SHORT-ORDINAL 000..111110

**Имя команды.** ZERO-ORDINAL 000..000001

**Имя команды.** ZERO-REAL 0..100001

**Имя команды.** ZERO-TEMPORARY-REAL 0..010001

**Выполняемая операция.** Операнду длиной 8, 16, 32, 64 или 80 бит присваивается значение, равное 0.

**Количество operandов.** 1

**Имя команды.** ONE-CHARACTER 01..011110

**Имя команды.** ONE-SHORT-ORDINAL 100..111110

**Имя команды.** ONE-ORDINAL 100..000001

**Выполняемая операция.** Операнду длиной 8, 16 или 32 бит присваивается значение, равное +1.

**Количество operandов.** 1

**Имя команды.** CONVERT-CHARACTER-TO-SHORT-ORDINAL  
001001

**Имя команды.** CONVERT-SHORT-ORDINAL-TO-CHARACTER

**Имя команды.** 01..101001

CONVERT-SHORT-ORDINAL-TO-ORDINAL  
00..011001

**Имя команды.** CONVERT-SHORT-ORDINAL-TO-TEMPORARY-  
REAL 0..111001

**Имя команды.** CONVERT-SHORT-INTEGER-TO-INTEGER  
10..011001

**Имя команды.** CONVERT-SHORT-INTEGER-TO-TEMPORARY-  
REAL 1..111001

**Имя команды.** CONVERT-ORDINAL-TO-SHORT-ORDINAL  
10..100101

**Имя команды.** CONVERT-ORDINAL-TO-INTEGER 1110..0100

**Имя команды.** CONVERT-ORDINAL-TO-TEMPORARY-REAL  
0..010101

**Имя команды.** CONVERT-INTEGER-TO-SHORT-INTEGER  
01..100101

**Имя команды.** CONVERT-INTEGER-TO-ORDINAL 1101..0100

**Имя команды.** CONVERT-INTEGER-TO-TEMPORARY-REAL  
01..010101

**Имя команды.** CONVERT-SHORT-REAL-TO-TEMPORARY-  
REAL 11..010101

**Имя команды.** CONVERT-REAL-TO-TEMPORARY-REAL 101101

**Имя команды.** CONVERT-TEMPORARY-REAL-TO-ORDINAL  
0..11101

**Имя команды.** CONVERT-TEMPORARY-REAL-TO-INTEGER  
01..111101

**Имя команды.** CONVERT-TEMPORARY-REAL-TO-SHORT-  
REAL 11..111101

**Имя команды.** CONVERT-TEMPORARY-REAL-TO-REAL 000011

*Выполняемая операция.* Данные, адресуемые первым операндом команды, преобразуются в соответствии с форматом данных, адресуемых вторым операндом, и размещаются на месте данных, адресуемых вторым операндом.

**Количество operandов.** 2

Если данные, адресуемые первым операндом команды, представлены в форме вещественного числа с плавающей точкой, а данные, адресуемые вторым операндом, имеют вид более короткого вещественного числа с плавающей точкой или представляют собой данные другого типа, выполняется округление, тип которого определяется состоянием флага управления округлением в контексте. Если данные, адресуемые вторым операндом — порядковое или короткое порядковое число, значение данных, адресуемых первым операндом, должно быть положительным. При выполнении данной команды могут возникнуть ошибки переполнения и антпереполнения.

## АРИФМЕТИЧЕСКИЕ КОМАНДЫ

**Имя команды.** INCREMENT-CHARACTER 001..110001

**Имя команды.** INCREMENT-SHORT-ORDINAL 1100..1000

**Имя команды.** INCREMENT-SHORT-INTEGER 1010..1000

**Имя команды.** INCREMENT-ORDINAL 010..0100

**Имя команды.** INCREMENT-INTEGER 0001..0100

*Выполняемая операция.* Данные, адресуемые первым операндом команды, увеличиваются на 1 и пересыпаются по адресу, указываемому вторым операндом.

**Количество operandов.** 2

**Имя команды.** DECREMENT-CHARACTER 101..110001

**Имя команды.** DECREMENT-SHORT-ORDINAL 0010..1000

**Имя команды.** DECREMENT-SHORT-INTEGER 0110..1000

**Имя команды.** DECREMENT-ORDINAL 0110..0100

**Имя команды.** DECREMENT-INTEGER 1001..0100

*Выполняемая операция.* Данные, адресуемые первым операн-

дом команды, уменьшаются на 1 и пересылаются по адресу, указываемому вторым операндом.

*Количество operandov. 2*

*Имя команды.* ADD-CHARACTER 001..010011

*Имя команды.* ADD-SHORT-ORDINAL 0110..0010

*Имя команды.* ADD-SHORT-INTEGER 01001..0010

*Имя команды.* ADD-ORDINAL 1010..0110

*Имя команды.* ADD-INTEGER 0101..0110

*Имя команды.* ADD-SHORT-REAL-SHORT-REAL 00..101011

*Имя команды.* ADD-SHORT-REAL-TEMPORARY-REAL  
00..010111

*Имя команды.* ADD-TEMPORARY-REAL-SHORT-REAL  
00..011011

*Имя команды.* ADD-TEMPORARY-REAL 00..101111

*Имя команды.* ADD-REAL-TEMPORARY-REAL 00..110111

*Имя команды.* ADD-TEMPORARY-REAL-REAL 00..100111

*Имя команды.* ADD-REAL-REAL 00..000111

*Выполняемая операция.* Данные, адресуемые двумя операндами команды, складываются и результат помещается по адресу, задаваемому третьим операндом.

*Количество operandov. 3*

Результатом выполнения всех команд над вещественными числами с плавающей точкой является длинное вещественное число с плавающей точкой. При формировании результата учитываются состояния флагов управления округлением и точностью в контексте. Некоторые команды над вещественными числами с плавающей точкой допускают использование данных различной точности, адресуемых первыми двумя операндами команды.

*Имя команды.* SUBTRACT-CHARACTER 101..010011

*Имя команды.* SUBTRACT-SHORT-ORDINAL 01110..0010

*Имя команды.* SUBTRACT-SHORT-INTEGER 11001..0010

*Имя команды.* SUBTRACT-ORDINAL 0110..0110

*Имя команды.* SUBTRACT-INTEGER 1101..0110

*Имя команды.* SUBTRACT-SHORT-REAL-SHORT-REAL  
10..101011

*Имя команды.* SUBTRACT-SHORT-REAL-TEMPORARY-REAL  
10..010111

*Имя команды.* SUBTRACT-TEMPORARY-REAL-SHORT-REAL  
10..011011

*Имя команды.* SUBTRACT-REAL-REAL 10..000111

*Имя команды.* SUBTRACT-REAL-TEMPORARY-REAL 10..110111

*Имя команды.* SUBTRACT-TEMPORARY-REAL-REAL 10..100111

*Имя команды.* SUBTRACT-TEMPORARY-REAL 10..101111

**Выполняемая операция.** Данные, адресуемые первым операндом команды, вычтитаются из данных, адресуемых вторым операндом. Результат помещается по адресу, задаваемому третьим операндом.

#### **Количество operandов. 3**

Результатом выполнения всех команд над вещественными числами с плавающей точкой является длинное вещественное число с плавающей точкой. При формировании результата учитываются состояние флагов управления округлением и точностью в контексте. Некоторые команды над вещественными числами с плавающей точкой допускают использование данных различной точности, адресуемых первыми двумя операндами команды.

**Имя команды.** MULTIPLY-SHORT-ORDINAL 11110..0010

**Имя команды.** MULTIPLY-SHORT-INTEGER 00101..0010

**Имя команды.** MULTIPLY-ORDINAL 1110..0110

**Имя команды.** MULTIPLY-INTEGER 0011..0110

**Имя команды.** MULTIPLY-SHORT-REAL-SHORT-REAL  
00..010111

**Имя команды.** MULTIPLY-SHORT-REAL-TEMPORARY-REAL  
01..010111

**Имя команды.** MULTIPLY-TEMPORARY-REAL-SHORT-REAL  
01..101011

**Имя команды.** MULTIPLY-REAL-REAL 01..000111

**Имя команды.** MULTIPLY-REAL-TEMPORARY-REAL 01..110111

**Имя команды.** MULTIPLY-TEMPORARY-REAL-REAL 01..100111

**Имя команды.** MULTIPLY-TEMPORARY-REAL 01..101111

**Выполняемая операция.** Результат умножения данных, адресуемых первыми двумя operandами команды, помещается по адресу, задаваемому третьим operandом.

#### **Количество operandов. 3**

Если перемножаются не вещественные числа, то по адресу третьего operandана помещаются младшие 16 или 32 бит результата. При выполнении данной команды переполнения не бывает. Результатом выполнения всех команд над вещественными числами с плавающей точкой является длинное вещественное число с плавающей точкой. При формировании результата учитываются состояния флагов управления округлением и точностью в контексте. Некоторые команды над вещественными числами с плавающей точкой допускают использование данных разной точности, адресуемых первыми двумя operandами команды.

**Имя команды.** DIVIDE-SHORT-ORDINAL 00001..0010

**Имя команды.** DIVIDE-SHORT-INTEGER 10101..0010

*Имя команды.* DIVIDE-ORDINAL 0001..0110

*Имя команды.* DIVIDE-INTEGER 1011..0110

*Имя команды.* DIVIDE-SHORT-REAL-SHORT-REAL 11..101011

*Имя команды.* DIVIDE-SHORT-REAL-TEMPORARY-REAL  
11..010111

*Имя команды.* DIVIDE-TEMPORARY-REAL-SHORT-REAL  
11..011011

*Имя команды.* DIVIDE-REAL-REAL 11..000111

*Имя команды.* DIVIDE-TEMPORARY-REAL-REAL 11..100111

*Имя команды.* DIVIDE-TEMPORARY-REAL 011..101111

*Имя команды.* DIVIDE-REAL-TEMPORARY-REAL 11..110111

*Выполняемая операция.* Производится деление данных, адресуемых вторым операндом команды, на данные, адресуемые первым операндом; результат помещается по адресу, задаваемому третьим операндом команды.

*Количество operandов.* 3

Если операция выполняется не над вещественными числами с плавающей точкой и делимое не является числом, кратным делителю, результат усекается с округлением в сторону нуля. В результате выполнения всех команд над вещественными числами с плавающей точкой получается длинное вещественное число с плавающей точкой. При формировании результата учитываются состояния флагов управления округлением и точностью в контексте. Некоторые команды над вещественными числами с плавающей точкой допускают использование данных различной точности, адресуемых первыми двумя операндами команды.

*Имя команды.* REMAINDER-SHORT-ORDINAL 10001..0010

*Имя команды.* REMAINDER-SHORT-INTEGER 01101..0010

*Имя команды.* REMAINDER-ORDINAL 1001..0110

*Имя команды.* REMAINDER-INTEGER 0111..0110

*Имя команды.* REMAINDER-TEMPORARY-REAL 10..100011

*Выполняемая операция.* Производится деление данных, адресуемых вторым операндом команды, на данные, адресуемые первым операндом; остаток помещается по адресу, задаваемому третьим операндом.

*Количество operandов.* 3

Для данных типа целое число или короткое целое число со знаком знак остатка совпадает со знаком делимого. В случае длинных вещественных чисел с плавающей точкой деление выполняется по итеративному алгоритму до заданного числа шагов или до получения частичного остатка, абсолютная величина которого не превышает величины делителя. В последнем случае частичный остаток является истинным результатом. При формировании результата учитываются состояния флагов

управления округлением и точностью в контексте. Если же частичный остаток не является еще истинным результатом, округление и установление требуемой точности не выполняются. Знак результата совпадает со знаком делимого.

**Имя команды.** ABSOLUTE-VALUE-SHORT-REAL 1011..0100

**Имя команды.** ABSOLUTE-VALUE-REAL 11..001101

**Имя команды.** ABSOLUTE-VALUE-TEMPORARY-REAL  
11..100011

**Выполняемая операция.** Абсолютное значение данных, адресуемых первым операндом команды, помещается по адресу, задаваемому вторым операндом.

**Количество operandов.** 2

**Имя команды.** NEGATE-SHORT-INTEGER 1110..1000

**Имя команды.** NEGATE-INTEGER 0101..0100

**Имя команды.** NEGATE-SHORT-REAL 0011..0100

**Имя команды.** NEGATE-REAL 10..100011

**Имя команды.** NEGATE-TEMPORARY-REAL 10..100011

**Выполняемая операция.** Данные, адресуемые первым операндом команды, с противоположным знаком помещаются по адресу, задаваемому вторым операндом.

**Количество operandов.** 2

**Имя команды.** SQUARE-ROOT-TEMPORARY-REAL 01..100011

**Выполняемая операция.** Из данных, адресуемых первым операндом команды, извлекается квадратный корень и помещается по адресу, задаваемому вторым операндом.

**Количество operandов.** 2

При формировании результата учитываются состояния флагов управления округлением и точностью в контексте.

## КОМАНДЫ СРАВНЕНИЯ

**Имя команды.** EQUAL-CHARACTER 0011..010011

**Имя команды.** EQUAL-SHORT-ORDINAL 000..1100

**Имя команды.** EQUAL-ORDINAL 000..1010

**Имя команды.** EQUAL-SHORT-REAL 011..1010

**Имя команды.** EQUAL-REAL 0..111011

**Имя команды.** EQUAL-TEMPORARY-REAL 0..001111

**Выполняемая операция.** Производится сравнение данных, адресуемых первыми двумя операндами команды. Результат сравнения в виде логического значения («истинно» — при равенстве сравниваемых данных и «ложно» — в противоположном случае) помещается по адресу, задаваемому операндом.

**Количество operandов.** 3

Третий operand адресует 1 байт символьных данных.

*Примечание.* Сравнение целых чисел и коротких целых чисел может выполняться с помощью команд для порядковых и коротких порядковых чисел.

**Имя команды.** EQUAL-ZERO-CHARACTER 011..110001

**Имя команды.** EQUAL-ZERO-SHORT-ORDINAL 00..101001

**Имя команды.** EQUAL-ZERO-ORDINAL 00..000101

**Имя команды.** EQUAL-ZERO-SHORT-REAL 101..000101

**Имя команды.** EQUAL-ZERO-REAL 0..110101

**Имя команды.** EQUAL-ZERO-TEMPORARY-REAL 0..011101

*Выполняемая операция.* Производится сравнение данных, адресуемых первым operandом команды с нулем. Результат сравнения в виде логического значения («истинно» — при равенстве данных нулю и «ложно» — в противоположном случае) помещается по адресу, задаваемому вторым operandом команды.

**Количество operandов.** 2

Второй operand адресует 1 байт символьных данных.

**Имя команды.** NOT-EQUAL-CHARACTER 1011..010011.

**Имя команды.** NOT-EQUAL-SHORT-ORDINAL 100..1100

**Имя команды.** NOT-EQUAL-ORDINAL 100..1010

*Выполняемая операция.* Производится сравнение данных, адресуемых первыми двумя operandами команды. Результат сравнения в виде логического значения («истинно» — при неравенстве данных и «ложно» — в противоположном случае) помещается по адресу, задаваемому третьим operandом.

**Количество operandов.** 3

Третий operand адресует 1 байт символьных данных.

**Имя команды.** NOT-EQUAL-ZERO-CHARACTER 111..110001

**Имя команды.** NOT-EQUAL-ZERO-SHORT-ORDINAL  
10..101001

**Имя команды.** NOT-EQUAL-ZERO-ORDINAL 010..000101

**Имя команды.** NOT-EQUAL-ZERO-SHORT-REAL 101..000101

**Имя команды.** NOT-EQUAL-ZERO-REAL 0..110101

**Имя команды.** NOT-EQUAL-ZERO-TEMPORARY-REAL  
0..011101

*Выполняемая операция.* Производится сравнение данных, адресуемых первым operandом команды, с нулем. Результат сравнения в виде логического значения («истинно» — при неравенстве нулю и «ложно» — в противоположном случае) помещается по адресу, задаваемому вторым operandом.

**Количество operandов.** 2

Второй operand адресует 1 байт символьных данных.

*Имя команды.* GREATER-THAN-CHARACTER 0111..010011

*Имя команды.* GREATER-THAN-SHORT-ORDINAL 010..1100

*Имя команды.* GREATER-THAN-SHORT-INTEGER 001..1100

*Имя команды.* GREATER-THAN-ORDINAL 010..1010

*Имя команды.* GREATER-THAN-INTEGER 001..1010

*Имя команды.* GREATER-THAN-SHORT-REAL 0111..1010

*Имя команды.* GREATER-THAN-REAL 01..111011

*Имя команды.* GREATER-THAN-TEMPORARY- REAL 01..001111

*Выполняемая операция.* Если значение данных, адресуемых вторым операндом команды, больше значения данных, адресуемых первым операндом, то по адресу, задаваемому третьим операндом, помещается логическое значение «истинно». В противном случае по адресу, задаваемому третьим операндом, располагается логическое значение «ложно».

*Количество operandов.* 3

Третий operand адресует 1 байт символьных данных.

*Имя команды.* GREATER-THAN-OR-EQUAL-CHARACTER  
1111..010011

*Имя команды.* GREATER-THAN-OR-EQUAL-SHORT-ORDINAL  
110..1100

*Имя команды.* GREATER-THAN-OR-EQUAL-SHORT-INTEGER  
101..1100

*Имя команды.* GREATER-THAN-OR-EQUAL-ORDINAL  
110..1010

*Имя команды.* GREATER-THAN-OR-EQUAL-INTEGER 101..1010

*Имя команды.* GREATER-THAN-OR-EQUAL-SHORT-REAL  
1111..1010

*Имя команды.* GREATER-THAN-OR-EQUAL-REAL 11..111011

*Имя команды.* GREATER-THAN-OR-EQUAL-TEMPORARY-  
REAL 11..001111

*Выполняемая операция.* Если значение данных, адресуемых вторым операндом команды, больше или равно значению данных, адресуемых первым операндом, то по адресу, задаваемому третьим операндом, помещается логическое значение «истинно». В противном случае по адресу, задаваемому третьим операндом, располагается логическое значение «ложно».

*Количество operandов.* 3

Третий operand адресует 1 байт символьных данных.

*Имя команды.* POSITIVE-SHORT-INTEGER 011..101001

*Имя команды.* POSITIVE-INTEGER 110..000101

*Имя команды.* POSITIVE-SHORT-REAL 011..000101

*Имя команды.* POSITIVE-REAL 01..110101

*Имя команды.* POSITIVE-TEMPORARY-REAL 01..011101

*Выполняемая операция.* Если данные, адресуемые первым опе-

рандом команды, больше нуля, то по адресу, задаваемому вторым операндом, помещается логическое значение «истинно». В противном случае по адресу, задаваемому вторым операндом, располагается логическое значение «ложно».

#### *Количество operandов. 2*

Второй operand адресует 1 байт символьных данных.

*Примечание.* Применительно к этой команде нуль не интерпретируется как положительное число.

*Имя команды.* NEGATIVE-SHORT-INTEGER 111..101001

*Имя команды.* NEGATIVE-INTEGER 001..000101

*Имя команды.* NEGATIVE-SHORT-REAL 111..000101

*Имя команды.* NEGATIVE-REAL 11..110101

*Имя команды.* NEGATIVE-TEMPORARY-REAL 11..011101

*Выполняемая операция.* Если значение данных, адресуемых первым operandом команды, меньше нуля, то по адресу, задаваемому вторым operandом, помещается логическое значение «истинно», в противном случае — значение «ложно».

#### *Количество operandов. 2*

Второй operand адресует 1 байт символьных данных.

### ЛОГИЧЕСКИЕ КОМАНДЫ

*Имя команды.* AND-CHARACTER 001..010011

*Имя команды.* AND-SHORT-ORDINAL 0000..0010

*Имя команды.* AND-ORDINAL 000..0110

*Выполняемая операция.* Над данными, адресуемыми первыми двумя operandами команды, поразрядно выполняется логическая операция конъюнкции (И). Результат помещается по адресу, задаваемому третьим operandом.

#### *Количество operandов. 3*

*Имя команды.* OR-CHARACTER 100..010011

*Имя команды.* OR-SHORT-ORDINAL 1000..0010

*Имя команды.* OR-ORDINAL 0100..0110

*Выполняемая операция.* Над данными, адресуемыми первыми двумя operandами команды, поразрядно выполняется логическая операция дизъюнкции (ИЛИ). Результат помещается по адресу, задаваемому третьим operandом.

#### *Количество operandов. 3*

*Имя команды.* XOR-CHARACTER 010..010011

*Имя команды.* XOR-SHORT-ORDINAL 0100..0010

*Имя команды.* XOR-ORDINAL 1100..0110

*Выполняемая операция.* Над данными, адресуемыми первыми двумя operandами команды, поразрядно выполняется логиче-

сская операция ИСКЛЮЧАЮЩЕЕ ИЛИ. Результат помещается по адресу, задаваемому третьим операндом.

*Количество operandов.* 3

*Имя команды.* XNOR-CHARACTER 110..010011

*Имя команды.* XNOR-SHORT-ORDINAL 1100..0010

*Имя команды.* XNOR-ORDINAL 0010..0110

*Выполняемая операция.* Над данными, адресуемыми первыми двумя операндами команды, поразрядно выполняется логическая операция НЕ-ИСКЛЮЧАЮЩЕЕ ИЛИ. Результат помещается по адресу, задаваемому третьим операндом.

*Количество operandов.* 3

*Имя команды.* COMPLEMENT-CHARACTER 10..110001

*Имя команды.* COMPLEMENT-SHORT-ORDINAL 1000..1000

*Имя команды.* COMPLEMENT-ORDINAL 100..0100

*Выполняемая операция.* Над данными, адресуемыми первым операндом команды, поразрядно выполняется логическая операция НЕ (отрицание). Результат помещается по адресу, задаваемому вторым операндом.

*Количество operandов.* 2

*Имя команды.* EXTRACT-SHORT-ORDINAL 0010..0010

*Имя команды.* EXTRACT-ORDINAL 00..001011

*Выполняемая операция.* Из кода данных, адресуемых вторым операндом команды, извлекается строка последовательно расположенных битов и помещается (с выравниванием по правой границе) по адресу, задаваемому третьим операндом.

*Количество operandов.* 3

Данные, адресуемые первым операндом команды, являются коротким порядковым числом. Наименьший значащий байт представления этого числа определяет порядковый номер первого бита извлекаемой последовательности битов. В зависимости от того, являются ли данные, адресуемые вторым и третьим операндами, порядковыми или короткими порядковыми числами, используются только младшие 4 или 5 бит этого байта. Порядковый номер первого бита извлекаемой последовательности определяет наименьший значащий бит этой последовательности, а также указывает его положение в виде смещения (по числу разрядов) от наименьшего значащего бита данных, адресуемых вторым операндом. Значение старшего байта определяет уменьшеннное на единицу число битов извлекаемой последовательности. С помощью указанных 2 байт может определяться последовательность, образующаяся многократным циклическим копированием (слева направо) битов данных, адресуемых вторым операндом.

Данные, адресуемые вторым операндом команды, являются порядковым или коротким порядковым числом, из которого извлекается последовательность битов. Выделенная последовательность помещается по адресу, задаваемому третьим операндом, который определяет порядковое или короткое порядковое число. Помещаемая последовательность выравнивается по правой границе поля с заполнением его свободной части ведущими нулями.

*Имя команды.* INSERT-SHORT-ORDINAL 1010..0010

*Имя команды.* INSERT-ORDINAL 10..001011

*Выполняемая операция.* Из двоичного кода данных, адресуемых вторым операндом команды, извлекается строка последовательно расположенных битов (начиная с младшего разряда справа) и помещается в область, задаваемую третьим операндом.

*Количество operandov.* 3

Эти команды выполняются подобно командам EXTRACT, за исключением того, что первый операнд наряду с длиной определяет и позицию начала размещения пересылаемой последовательности в области, адресуемой третьим операндом команды.

*Примечание.* Существуют разновидности этих команд для неделимого выполнения операций<sup>1)</sup>.

*Имя команды.* SIGNIFICANT-BIT-SHORT-ORDINAL  
0100..1000

*Имя команды.* SIGNIFICANT-BIT-ORDINAL 00..100101

*Выполняемая операция.* Определяется положение старшего бита в двоичном коде данных, адресуемых первым операндом команды. Порядковый номер этого разряда помещается по адресу, задаваемому вторым операндом.

*Количество operandov.* 2

Данные, адресуемые первым операндом команды, представляют собой порядковое или короткое порядковое число; данные, адресуемые вторым операндом — короткое порядковое число.

## КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

*Имя команды.* CALL-CONTEXT 101..000001

*Выполняемая операция.* Выполнение команд текущего контекста приостанавливается. Создается объект «контекст»; управление

<sup>1)</sup> Эти разновидности не включены в данное описание. Особенности неделимого выполнения команд, т. е. выполнения без прерывания, обсуждаются в гл. 17 и далее на примере команд сложения порядковых и коротких порядковых чисел (см., например, команду INDIVISIBLY-ADD-SHORT-ORDINAL). — Прим. перев.

передается определенному сегменту команд, адресуемому через задаваемый домен.

#### **Количество operandов. 2**

Первый операнд команды адресуется к дескриптору доступа к объекту «домен». Второй операнд — адрес короткого порядкового числа, которое используется в качестве индекса для нахождения дескриптора к сегменту команд в выделенном домене.

#### **Имя команды. CALL-CONTEXT-WITH-MESSAGE 111..100101**

**Выполняемая операция.** Выполнение команд текущего контекста приостанавливается. Создается новый объект «контекст». В созданный контекст помещается дескриптор доступа к специальному указываемому объекту. Управление передается определенному сегменту команд, адресуемому через задаваемый домен.

#### **Количество operandов. 3**

Первым операндом является дескриптор доступа к объекту «домен». Второй operand — короткое порядковое число, используемое в качестве индекса для нахождения дескриптора доступа к сегменту команд в выделенном домене. Третий operand — дескриптор доступа, поставляемый в новый контекст, т. е. дескриптор доступа к объекту «сообщение».

#### **Имя команды. RETURN ..001110**

**Выполняемая операция.** Уничтожается текущий объект «контекст», а также все объекты, имеющие в таблице объектов процесса номера уровня больше или равные номеру уровня данного контекста. Управление передается команде, адресуемой посредством указателя команд в контексте, определяемом содержимым поля «дескриптор доступа предыдущего контекста» в уничтожаемом контексте.

#### **Количество operandов. 0**

**Примечание.** При выполнении команды обновляется содержимое дескриптора блока памяти для текущего процесса, что обеспечивает своевременную регистрацию освобождаемой памяти.

#### **Имя команды. BRANCH ..101110**

**Выполняемая операция.** Управление передается определенной команде в пределах данного сегмента команд.

#### **Количество operandов. 1 (адрес перехода).**

#### **Имя команды. BRANCH-INDIRECT 001..111110**

**Выполняемая операция.** Управление передается команде в пределах данного сегмента команд, смещение которой в битах задается значением операнда в форме короткого порядкового числа.

#### **Количество operandов. 1**

**Имя команды.** BRANCH-INTERSEGMENT 110..000001

**Выполняемая операция.** Управление передается заданий команде в указанном сегменте команд.

**Количество operandов.** 1

Поле длиной 32 бит, адресуемое операндом, используется для записи двух коротких порядковых чисел. Первое из этих чисел — индекс в домене, связанным с текущим контекстом. Этот индекс определяет дескриптор доступа к сегменту команд. Второе число задает (в битах) смещение команды.

**Имя команды.** BRANCH-INTERSEGMENT-WITHOUT-TRACE 001..000001

**Выполняемая операция.** Производятся те же действия, что и по команде BRANCH-INTERSEGMENT с тем отличием, что в данном случае ошибки трассировки на выполнение команды не влияют.

**Количество operandов.** 1

**Имя команды.** BRANCH-INTERSEGMENT-AND-LINK 1111..0100

**Выполняемая операция.** По адресу, задаваемому вторым операндом, формируется адрес следующей команды. Управление передается определенной команде в указанном сегменте команд.

**Количество operandов.** 2

Содержимое 32-битовых полей, адресуемых операндами команды, воспринимается как четыре коротких порядковых числа. Первое используется в качестве индекса в домене, связанным с текущим контекстом. Посредством этого индекса определяется дескриптор доступа к сегменту команд. Второе число определяет смещение (в битах) команды в сегменте. Третье число определяет индекс текущего сегмента команд в домене, четвертое — смещение в битах для команды, непосредственно следующей за данной.

**Имя команды.** BRANCH-TRUE 0..0000

**Выполняемая операция.** Если операнд типа «символьные данные» принимает логическое значение «истинно», управление передается адресуемой команде в пределах данного сегмента команд. В противном случае управление передается следующей команде.

**Количество operandов.** 2 (короткое порядковое число и адрес перехода).

**Имя команды.** BRANCH-FALSE 0..0000

**Выполняемая операция.** Если операнд типа «символьные данные» принимает логическое значение «ложно», управление пе-

редается адресуемой команде в данном сегменте команд. В противном случае управление передается следующей команде. *Количество operandов.* 2 (короткое порядковое число и адрес перехода).

*Имя команды.* SET-CONTEXT-MODE 011..111110

*Выполняемая операция.* В поле текущего объекта «контекст», определяющее состояние контекста, помещается содержимое поля, адресуемого операндом (короткое порядковое число).

*Количество operandов.* 1

*Примечание.* Эта операция позволяет изменять значение флага управления обработкой неточных результатов, флага задания точности и флагов управления округлением и обработкой ошибок в контексте.

## КОМАНДЫ АДРЕСАЦИИ

*Имя команды.* CREATE-DATA-SEGMENT 10011..0010

*Выполняемая операция.* Распределяется память для сегмента данных требуемой длины. Эта память выделяется либо из указываемого объекта «ресурсы памяти», либо из локальных ресурсов памяти текущего процесса. В таблицу объектов вносится соответствующий дескриптор объекта. При обращении за памятью в объект «ресурсы памяти» запись вносится в таблицу объектов, связанную с объектом «ресурсы памяти», в противном случае — в таблицу объектов, связанную с текущим процессом. Дескриптор доступа к сегменту сохраняется в указываемом сегменте доступа.

*Количество operandов.* 3

Первый операнд — короткое порядковое число, определяющее в байтах длину сегмента. Второй операнд — адрес объекта «ресурсы памяти» или адрес локальных ресурсов памяти данного процесса, если селектор объекта равен иулю. В третьем операнде задается адрес дескриптора доступа к формируемому сегменту памяти. В этом дескрипторе флаги прав записи и чтения устанавливаются в 1, флаги прав удаления устанавливаются в 0, а состояние хип-флага зависит от вида второго операнда.

*Примечание.* При выполнении всех команд, производящих запись в сегмент доступа, возникает ошибка типа «стирание дескриптора доступа, флаги удаления которых не сброшены», если в соответствующем поле сегмента доступа находился дескриптор доступа с установленными флагами удаления.

*Имя команды.* CREATE-ACCESS-SEGMENT 01011..0010

*Выполненная операция.* Распределяется память для сегмента

доступа требуемого размера. Эта память выделяется либо из указываемого объекта «ресурсы памяти», либо из локальных ресурсов памяти текущего процесса. В таблицу объектов вносится соответствующий дескриптор объекта. При обращении за памятью в объект «ресурсы памяти» запись вносится в таблицу объектов, связанную с объектом «ресурсы памяти», в противном случае — в таблицу объектов, связанную с текущим процессом. Дескриптор доступа к сегменту сохраняется в заданном сегменте доступа.

**Количество operandов.** См. описание команды CREATE-DATA-SEGMENT.

**Имя команды.** CREATE-TYPED-SEGMENT 01..110011

**Выполняемая операция.** Формируется сегмент, тип которого определяется в объекте «управление дескриптором». Память выделяется из указываемого объекта «ресурсы памяти» либо из локальных ресурсов памяти текущего процесса. В таблицу объектов вносится соответствующий дескриптор объекта. При обращении за памятью в объект «ресурсы памяти» запись вносится в таблицу объектов, связанную с объектом «ресурсы памяти», в противном случае — в таблицу объектов, связанную с текущим процессом. Дескриптор доступа к сегменту сохраняется в заданном сегменте доступа.

**Количество operandов.** 4

Первый, второй и третий операнды такие же, как и в команде CREATE-DATA-SEGMENT. В дескрипторе доступа устанавливаются все системные права доступа к сегменту. Четвертый операнд содержит обращение к объекту «управление дескриптором».

**Имя команды.** CREATE-GENERIC-REFINEMENT 01..001011

**Выполняемая операция.** В таблице объектов создается дескриптор аффинажа, содержащий описание части сегмента. Информация о типе для части сегмента идентична соответствующей информации для сегмента в целом. Дескриптор доступа к аффинажу запоминается в заданном сегменте доступа.

**Количество operandов.** 5

Первый операнд указывает адрес объекта «ресурсы памяти» или локальных ресурсов памяти текущего процесса, чем определяется используемая таблица объектов. Второй операнд задает положение дескриптора доступа к аффинажу. Третьим операндом является селектор дескриптора доступа к сегменту. Четвертый операнд представляет собой короткое порядковое число, определяющее длину аффинажа (части сегмента). Пятый операнд — короткое порядковое число, задающее смещение аффинажа в сегменте.

**Имя команды.** CREATE-TYPED-REFINEMENT 1111..0110

**Выполняемая операция.** В таблице объектов создается дескриптор аффинажа, содержащий описание части сегмента. Значение базового типа в дескрипторе такое же, как и у всего сегмента, а значение системного типа определяется объектом «управления аффинажем». Дескриптор доступа к аффинажу запоминается в указываемом сегменте доступа.

**Количество operandов.** 6

Первый operand является адресом объекта «управление аффинажем». Остальные operandы идентичны operandам команды CREATE-GENERIC-REFINEMENT.

**Имя команды.** CREATE-PRIVATE-TYPE 00..110011

**Выполняемая операция.** В таблице объектов создается дескриптор типа, устанавливающий связь между объектом и объектом «определение типа». Дескриптор доступа к дескриптору типа (субъекту расширенного типа) сохраняется в указываемом сегменте доступа.

**Количество operandов.** 4

Первый operand является адресом объекта «определение типа». Остальные три operandы совпадают с первыми тремя operandами команды CREATE-GENERIC-REFINEMENT.

**Имя команды.** CREATE-PUBLIC-TYPE 10..110011

**Выполняемая операция.** Производятся те же действия, что и по команде CREATE-PRIVATE-TYPE, однако дескриптор доступа помечается как «общий», а не «частный».

**Количество operandов.** 4

**Имя команды.** CREATE-ACCESS-DESCRIPTOR 110011..0010

**Выполняемая операция.** В указываемый сегмент доступа помещается дескриптор доступа к определенному дескриптору в таблице объектов.

**Количество operandов.** 3

Первый operand является адресом таблицы объектов, второй — адресом короткого порядкового числа, используемого как индекс для таблицы объектов. Третий operand представляет собой адрес сегмента доступа, куда будет помещен сформированный дескриптор доступа.

**Примечание.** Данная команда формирует потенциальный адрес к существующему сегменту. Для ее успешного выполнения необходимо иметь дескриптор доступа к таблице объектов с соответствующими правами доступа.

**Имя команды.** RETRIEVE-PUBLIC-TYPE-REPRESENTATION 0101..1000

**Выполняемая функция.** В указанный сегмент доступа помещается дескриптор доступа к объекту расширенного типа (адресуемому через дескриптор типа) при условии, что этот объект определен как объект общего пользования.

**Количество operandов.** 2

Первый операнд является селектором дескриптора доступа к объекту как объекту расширенного типа, т. е. дескриптора доступа, ссылающегося на дескриптор типа. Вторым operandом служит селектор дескриптора доступа к сегменту доступа, куда должен быть помещен результат.

**Имя команды.** RETRIEVE-TYPE-REPRESENTATION

11101..0010

**Выполняемая функция.** В указанный сегмент доступа помещается дескриптор доступа к объекту расширенного типа.

**Количество operandов.** 3

Первый операид является селектором дескриптора доступа к объекту как объекту расширенного типа, т. е. дескриптора доступа, ссылающегося на дескриптор типа. Вторым operandом служит селектор дескриптора доступа к объекту «определение типа». Третий operand — селектор дескриптора доступа к сегменту доступа, куда будет помещен результат. Система регистрирует ошибку, если адресуемый объект «определение типа» отличается от указываемого объекта расширенного типа.

**Имя команды.** RETRIEVE-TYPE-DEFINITION 1101..1000

**Выполняемая операция.** В указанный сегмент доступа помещается дескриптор доступа к объекту «определение типа», связанному с заданным дескриптором типа.

**Количество operandов.** 2

Первый операид является селектором дескриптора доступа к объекту расширенного типа, т. е. дескриптора доступа, ссылающегося на дескриптор типа. Второй operand представляет собой селектор дескриптора доступа к сегменту доступа, куда должен быть помещен результат.

**Примечание.** В первом ДД должны быть определены системные права на выполнение этой операции.

**Имя команды.** RETRIEVE-REFINED-OBJECT 00011..0010

**Выполняемая операция.** В указанный сегмент доступа помещается дескриптор доступа к сегменту, которому принадлежит адресуемый аффинаж.

**Количество operandов.** 3

Первый operand является селектором дескриптора доступа к объекту «управление аффинажем». Второй операид представ-

ляет собой селектор дескриптора доступа к аффинажу, а третий — селектор дескриптора доступа к сегменту доступа, куда должен быть помещен результат.

**Имя команды.** ENTER-ACCESS-SEGMENT 0111..1000

**Выполняемая операция.** Формируется значение ВСД текущего контекста, равное значению адресуемого сегмента доступа.

**Количество operandов.** 2

Первый operand является селектором дескриптора доступа к сегменту доступа. Второй operand представляет собой короткое порядковое число, принимающее значение 1, 2 или 3. Значение дескриптора доступа, адресуемого первым operandом, помещается в объект «контекст» в позицию ВСД с номером, определяемым вторым operandом.

**Имя команды.** ENTER-GLOBAL-ACCESS-SEGMENT 101..111110

**Выполняемая операция.** Сегмент доступа, адресуемый третьим дескриптором доступа в сегменте доступа процесса, назначается в качестве ВСД текущего контекста.

**Количество operandов.** 1

Operand адресует короткое порядковое число, как в команде ENTER-ACCESS-SEGMENT

**Имя команды.** COPY-ACCESS-DESCRIPTOR 0001..1000

**Выполняемая операция.** Значение дескриптора доступа, адресуемого первым operandом, переносится в качестве значения для дескриптора доступа, определяемого вторым operandом. В скопированной записи сбрасывается бит права на удаление.

**Количество operandов.** 2

**Примечание.** Как и в остальных аналогичных случаях, если в дескрипторе доступа, куда должен быть помещен результат, флаги удаления установлены в 1, возникает ошибка типа «стирание дескрипторов доступа, флаги удаления которых не сброшены».

**Имя команды.** NULL-ACCESS-DESCRIPTOR 110..111110

**Выполняемая операция.** В адресуемом дескрипторе доступа сбрасывается флаг готовности.

**Количество operandов.** 1

**Имя команды.** INSPECT-ACCESS-DESCRIPTOR 01..011001

**Выполняемая операция.** В сегменте данных сохраняется побитовая структура адресуемого дескриптора доступа.

**Количество operandов.** 2

**Имя команды.** INSPECT-ACCESS 011..011001

Для заданного дескриптора доступа производится выдача в спределенном формате информации (например, о правах, типах) об объекте, адресуемом этим дескриптором доступа.

**Количество operandов.** 2

**Имя команды.** RESTRICT-RIGHTS 011..100101

**Выполняемая операция.** Данная команда ограничивает права доступа в дескрипторе доступа.

**Количество operandов.** 2

Первый operand адресует порядковое число, рассматриваемое как объект «управление дескриптором». Второй operand является селектором дескриптора доступа, права доступа в котором должны быть изменены. Новые значения битов прав объекта, прав системы и хип-флага в дескрипторе доступа формируются как дизъюнкция (логическое ИЛИ) соответствующих значений в исходном дескрипторе доступа и в объекте «управление дескриптором». В результате права доступа либо сохраняются прежними, либо ограничиваются. Если в объекте «управление дескриптором» установлен флаг проверки типа, а значения типов объектов в объекте «управление дескриптором» и в адресуемом дескриптором доступа сегменте не совпадают, то регистрируется ошибка.

**Имя команды.** AMPLIFY-RIGHTS 1001..1000

**Выполняемая операция.** Данная команда расширяет права доступа в дескрипторе доступа на основании данных из объекта «управление дескриптором».

**Количество operandов.** 2

Первый operand адресует объект «управление дескриптором», а второй — дескриптор доступа, права доступа в котором должны быть изменены. Из объекта «управление дескриптором» в дескриптор доступа переносятся значения битов прав объекта, системы и хип-флага. Если в объекте «управление дескриптором» флаг проверки типа установлен в 1, а значения типов в этом объекте и в адресуемом дескриптором доступа сегменте не совпадают, то регистрируется ошибка.

**Примечание.** Первый operand данной команды в отличие от команды RESTRICT-RIGHTS должен быть адресной ссылкой на действительный объект «управление дескриптором».

## КОМАНДЫ, СВЯЗАННЫЕ С УПРАВЛЕНИЕМ ПРОЦЕССАМИ И ПРОЦЕССОРАМИ

**Имя команды.** SEND 01111..1000

**Выполняемая операция.** Объект (сообщение) ставится в очередь к порту в соответствии с установленными для него пра-

вилами обслуживания. Если память в очереди порта исчерпана или отсутствует, транспортер процесса ставится в очередь, а выполнение процесса приостанавливается до тех пор, пока не будет выделена требуемая память для очереди.

#### *Количество operandов. 2*

Первый operand адресует объект «порт». Второй может адресовать любой объект через дескриптор доступа.

#### *Имя команды. CONDITIONAL-SEND 10111..0010*

*Выполняемая операция.* Объект (сообщение) ставится в очередь к порту в соответствии с установленными для порта правилами обслуживания, если имеется свободная память в очереди. Символьной переменной, на которую ссылается один из operandов команды, присваивается логическое значение «истинно». В противном случае ни объект, ни носитель процесса в очередь не ставятся, а указанной символьной переменной присваивается логическое значение «ложно».

#### *Количество operandов. 3*

Первый operand адресует объект «порт». Второй может адресовать любой объект через дескриптор доступа. Третий operand адресует символьные данные длиной 1 байт.

#### *Имя команды. SURROGATE-SEND 11..110011*

*Выполняемая операция.* Объект (сообщение) посредством транспортера-заменителя ставится в очередь к порту в соответствии с установленными для порта правилами обслуживания.

#### *Количество operandов. 4*

Первый operand адресует первый (вспомогательный) порт, второй — через дескриптор доступа может произвести обращение к любому объекту. Третий operand адресует транспортер, а четвертый — еще один порт, куда должно быть переслано сообщение. Когда второй порт располагает достаточным местом для сообщения, транспортер автоматически пересыпается к нему.

*Примечание.* Команда выполняется аналогично команде SEND, за исключением того, что отсутствует риск возможной приостановки выполнения процесса. Используемый здесь транспортер должен быть независимым объектом (т. е. не транспортером данного процесса).

#### *Имя команды. RECEIVE 0111..111110*

*Выполняемая операция.* Из очереди к указанному порту удаляется первый объект (сообщение). Дескриптор доступа к этому объекту помещается в транспортер процесса. Если очередь пуста, транспортер процесса ставится в эту очередь и выполнение процесса приостанавливается до получения сообщения.

**Количество operandов.** 1 (адрес порта).

**Имя команды.** CONDITIONAL-RECEIVE 11111..1000

**Выполняемая операция.** Из очереди к указанному порту (если она не пуста) исключается первый объект (сообщение). Дескриптор доступа к этому объекту помещается в транспортер процесса. По адресу символьного операида помещается логическая величина «истинно». Если же очередь пуста, то по указанному адресу помещается величина «ложно».

**Количество operandов.** 2

Первый операид адресует объект «порт», второй — символьные данные длиной 1 байт.

**Имя команды.** SURROGATE-RECEIVE 111..011001

**Выполняемая операция.** Из очереди к указанному порту (если она не пуста) исключается первый объект (сообщение). Дескриптор доступа к этому объекту помещается в транспортер-заменитель, который пересыпается к другому порту. Если же очередь пуста, то указанный транспортер ставится в очередь на получение данных из этого порта.

**Количество operandов.** 3

Первый операид адресует объект «порт», второй — объект «транспортер», а третий — еще один объект «порт».

**Имя команды.** DELAY 1111..111110

**Выполняемая операция.** Выполнение процесса приостанавливается на указанный интервал времени. Для этого процесс ставится в очередь к специальному порту диспетчеризации, называемому портом задержки (дескриптор доступа к этому порту находится в объекте «процессор»).

**Количество operandов.** 1

Операид является адресом короткого порядкового числа. Значение этого числа является величиной требуемой временной задержки, исчисляемой в единицах времени внешнего тактового генератора (одна временная единица равна 100 мкс).

**Имя команды.** READ-PROCESS-CLOCK 011..000001

**Выполняемая операция.** Суммарное значение времени, в течение которого процессор выполнял операции данного процесса, помещается в виде порядкового числа по адресу, задаваемому операндом команды.

**Количество operandов.** 1

**Имя команды.** READ-PROCESSOR-STATUS-AND-CLOCK  
111..000001

**Выполняемая операция.** Две 16-битовые величины — системное

время процесса и состояние процессора — помещаются как порядковое число по адресу, задаваемому операндом команды.

#### *Количество operandов. 1*

*Примечание.* Информация о состоянии процессора содержится в той же форме, как и информация в объекте «процессор» (см. гл. 17), хотя и извлекается непосредственно из соответствующих микросхем.

#### *Имя команды. SEND-TO-PROCESSOR 0111..1100*

*Выполняемая операция.* Блокируется сегмент локальных связей процессора, которому должно быть послано сообщение. Анализируется значение в поле счетчика откликов в этом сегменте: если это значение не равно нулю, сегмент деблокируется и адресуемой символьной переменной присваивается значение «ложно», в противном случае в сегмент связи копируется значение определенного допустимого межпроцессорного сообщения. Значение в поле счетчика откликов в этом сегменте устанавливается равным 1, сегмент деблокируется, адресуемой символьной переменной присваивается значение «истинно» и на шину памяти выставляется межпроцессорный сигнал с идентификатором процессора-получателя.

#### *Количество operandов. 3*

Первый операнд адресует процессор-получатель (его объект «процессор»), второй операнд — короткое порядковое число, имеющее вид соответствующего межпроцессорного сообщения. Например, как отмечалось в гл. 17, такими сообщениями могут быть: «Пуск процессора», «Войти в режим реконфигурации» и т. п. Третий операнд адресует символьные данные длиной 1 байт.

#### *Имя команды. BROADCAST-TO-PROCESSORS 1111..1100*

*Выполняемая операция.* Блокируется сегмент глобальных связей. Анализируется значение в поле счетчика откликов в этом сегменте: если это значение не равно нулю, сегмент деблокируется и адресуемой символьной переменной присваивается значение «ложно»; в противном случае в сегмент связи копируется значение определенного допустимого межпроцессорного сообщения. В поле счетчика откликов в этом сегменте помещается значение из его поля счетчика процессоров, сегмент деблокируется, адресуемой символьной переменной присваивается значение «истинно» и на шину памяти поступает межпроцессорный сигнал с идентификатором 0 (обозначающим обращение ко всем процессорам).

#### *Количество operandов. 3*

Первый операнд адресует любой объект «процессор», сегмент доступа которого содержит обращение к глобальному сег-

менту доступа. Два остальных операнда такие же, как в команде SEND-TO-PROCESSOR

**Имя команды.** MOVE-TO-INTERCONNECT 01111..0010

**Выполняемая операция.** Осуществляется пересылка данных в несколько системных регистров, не входящих в процессор, или из подобных регистров. Назначение этих регистров зависит от конфигурации системы и здесь обсуждению не подлежит.

## КОМАНДЫ СИНХРОНИЗАЦИИ

**Имя команды.** LOCK-OBJECT 011..1100

**Выполняемая операция.** Если указываемая блокировка не установлена (два младших бита равны нулю), она устанавливается (упомянутым битам присваивается значение 10), а адресуемой символьной переменной присваивается значение «истинно». В противном случае указанной символьной переменной присваивается значение «ложно».

**Количество operandов.** 3

Первый operand адресует сегмент данных, второй — короткое порядковое число, значение которого используется в качестве смещения 16-битового объекта «блокировка» внутри сегмента. Третий operand адресует символьные данные длиной 1 байт.

**Примечание.** Поскольку при выполнении команды имеет место модификация сегмента, в дескрипторе доступа к нему должны иметься права записи. На время между анализом того, имеется блокировка или нет, и ее установлением шина памяти захватывается монопольно.

**Имя команды.** UNLOCK-OBJECT 0011..1000

**Выполняемая операция.** Деблокировка (двум младшим битам присваивается значение 00).

**Количество operandов.** 2

Операнды подобны первым двум operandам команды LOCK-OBJECT.

**Имя команды.** INDIVISIBLY-ADD-SHORT-ORDINAL 1011..1000

**Имя команды.** INDIVISIBLY-ADD-ORDINAL 0111..0100

**Выполняемая операция.** Суммируются значения двух operandов. На время от выборки operandов до записи результата шина памяти захватывается монопольно.

**Количество operandов.** 2

## ПОРЯДОК РАБОТЫ СО СТЕКОМ

Как уже отмечалось, один или несколько операндов команды могут адресоваться к данным, находящимся в стеке. Кроме того, стек может использоваться и для формирования адресных ссылок при обращении к данным. Вследствие этого следует уделять особое внимание последовательности расположения данных в стеке.

Наиболее естественный способ получения представления этой последовательности состоит в описании порядка, в котором обработка команды выполняется аппаратными средствами. При этом следует помнить, что извлечение данных из стека происходит всякий раз, когда к стеку производится обращение за адресом или данными. Единственное исключение составляет копирование значения содержимого стека по команде SAVE.

Машина обрабатывает операнды последовательно: первый операнд, второй и т. д. При обработке каждого операнда вначале формируется адрес, а затем извлекаются данные (выборка данных осуществляется до начала обработки следующего операнда). При формировании адреса с помощью стека, компоненты адреса выбираются в следующем порядке: база, индекс, затем селектор операнда.

## АССОЦИАТИВНОЕ ПРЕОБРАЗОВАНИЕ АДРЕСОВ АППАРАТНЫМИ СРЕДСТВАМИ

Архитектуре общего процессора данных системы iAPX 432 свойственна определенная избыточность (проявляющаяся, например, в многочисленных обращениях к памяти) при выполнении даже простых операций. Однако степень избыточности на самом деле значительно сокращается благодаря некоторым схемным особенностям реализации системы. Отметим основные из них.

1. Часть наиболее часто обновляемых данных из объектов «процессор» и «процесс», такие, как время процессора, время процесса или указатель команд, может сниматься непосредственно с соответствующих элементов микросхем. Во время активного состояния процессоров и процессов эта информация в соответствующих объектах может оказаться неточной.

2. Во время обработки процессором процесса в заданном контексте часть данных, связанных с адресацией, переводится в физические адреса и сохраняется в микросхемах процессора. К этим данным относятся: физические адреса сегментов доступа и сегментов данных, связанных с объектами «процессор», «процесс», «контекст», «порт диспетчеризации», «справочник

таблиц объектов», а также физические адреса и длины ВСД контекста.

3. Для размещения адресных ссылок (выборки данных по указываемым адресам) в микросхемах процессора предусмотрено два блока ассоциативной памяти.

В первом блоке ассоциативной памяти содержится информация об адресах четырех сегментов данных, к которым выполнялись последние обращения. Благодаря этой информации процессор может непосредственно переходить от значения селектора объекта в адресе данных к необходимому сегменту в реальной памяти, не обращаясь к контексту, ВСД, справочнику таблиц объектов и таблице объектов. В каждой из четырех записей этого блока памяти содержится информация, извлекаемая из адреса данных, дескриптора доступа и дескриптора объекта, определявшихся при нахождении соответствующего сегмента при последнем обращении к сегменту, когда данные о нем еще не находились в ассоциативной памяти. Эта информация включает селектор объекта (номер уровня ВСД и индекс ВСД), физический или «базовый» адрес, длину сегмента, флаг изменения, права чтения/записи.

Код селектора объекта представляет собой исходные данные для поиска объекта, выполняемого процессором. Если в ассоциативной памяти находится нужный код, то процессор сразу получает физический адрес сегмента. Если же требуемого кода селектора в памяти нет, выполняется обычный процесс адресации и параметры, определенные в процессе вычисления адреса, запоминаются вместо «наиболее старой» записи в блоке ассоциативной памяти.

Как отмечено выше, в число параметров, запоминаемых в записях ассоциативной памяти, включен флаг изменения. Это связано с тем, что, если значение этого флага равно 0 и командой предусматривается запись в сегмент данных, то ограничиться данными ассоциативной памяти невозможно и процессор должен провести обычный процесс адресации и найти дескриптор объекта, чтобы значение флага изменения в нем установить в 1.

Другим блоком ассоциативной памяти является буфер справочника таблиц объектов. В нем сохраняется информация о двух элементах справочника таблиц объектов, к которым выполнялись последние обращения. Это повышает эффективность адресации в случаях, когда не удается отыскать нужный код в первом блоке ассоциативной памяти. Новая запись в буфер заносится каждый раз, когда в процессе адресации требуется обращение к справочнику таблиц объектов. Эта запись включает следующую информацию: индекс справочника, базовый ад-

рес (физический адрес таблицы объектов); длину таблицы объектов; флаг изменения.

Значение индекса справочника используется, в частности, процессором при установлении адресации через справочник таблиц объектов.

На основании вышеизложенного можно было заметить, что в рассматриваемой мультипроцессорной системе возможны состояния, когда изменение данных в памяти одним из процессоров приведет к наличию некорректной информации в буферах адресации в другом процессоре. Примером может служить установление признака неготовности в дескрипторе некоторого объекта определенным процессором (например, операционной системой) одного из процессоров, в то время как какой-либо из процессов другого процессора продолжает пользоваться соответствующим сегментом. То же может произойти и с другими данными (имеющими отношение, например, к процессам или контекстам), хранящимися и выбираемыми непосредственно из микросхем процессоров.

Эти сложности преодолеваются с помощью средств передачи сообщений бездействующим процессорам, рассмотренных в гл. 17. Когда процессор обнаруживает состояния, при которых хранящиеся и выбираемые непосредственно из микросхем других процессоров данные оказываются недействительными (например, устаревшими вследствие изменений, внесенных в таблицу объектов), он автоматически посылает соответствующее оповещающее сообщение всем остальным процессорам. Функции обмена устаревших значений параметров, хранящихся непосредственно микросхемами процессоров, предоставляются также и программам (например, операционной системе) командами SEND-TO-PROCESSOR и BROADCAST-TO-PROCESSORS. Для этого могут использоваться следующие типы сообщений бездействующим процессорам:

1. «Сделать недействительной информацию в кэш-памяти сегмента данных», что вызовет очистку принимающими процессорами первого из указанных выше буферов адресации.

2. «Сделать недействительной информацию в кэш-памяти таблицы объектов», что вызовет очистку принимающими процессорами обоих буферов адресации.

3. «Приостановить выполнение и установить заново информацию контекста», что, как и предыдущее сообщение, вызовет очистку обоих буферов, а также обновление процессорами хранящейся в их микросхемах и связанный с контекстами информации по данным текущего объекта «контекст».

4. «Приостановить выполнение и установить заново информацию процесса», что приведет к реализации всех упомянутых выше действий, а также вызовет обновление принимающими

процессорами хранимой в их микросхемах информации, связанной с объектами «процесс».

5. «Приостановить работу и установить заново информацию процессора», что приведет к реализации всех упомянутых выше действий, а также вызовет обновление принимающими процессорами хранимой в их микросхемах информации, связанной с объектами «процессор».

6. «Приостановить работу и установить заново всю информацию процессора», что приведет к реализации всех упомянутых выше действий, а также вызовет полную начальную установку процессора при очередном запуске системы.

## УПРАЖНЕНИЯ

18.1. В приведенном наборе команд отсутствует команда EQUAL-INTEGER<sup>1)</sup>. Для сравнения целых чисел можно пользоваться командой EQUAL-ORDINAL, поскольку и целые, и порядковые числа представляются 32-битовыми значениями и для проверки на равенство достаточно провести побитовое сравнение (отсутствует необходимость в специальной обработке знака числа). Почему тогда для сравнения вещественных чисел разработана специальная команда EQUAL-REAL, хотя вещественные числа представляются также 32 бит?

18.2. В целях практики формирования адресных ссылок к данным расшифруйте код следующей команды (КОП расположен слева):

0111100000000000000000001111111001100000000100110100001001000011000000110

*Пояснение.* Расшифровку выполняйте справа налево. Данная команда является командой сравнения.

---

<sup>1)</sup> То есть команда сравнения целых чисел. — Прим. перев.

---

## ЧАСТЬ VII

# АРХИТЕКТУРА БАЗЫ ДАННЫХ

### ГЛАВА 19

#### СИСТЕМЫ С АССОЦИАТИВНОЙ ПАМЯТЬЮ

Как упоминалось в гл. 2, традиционным машинам присущи разнообразные по типу семантические разрывы между их архитектурой и языками программирования. Один из них связан с проблемами записи и чтения информации долговременного хранения, т. е. с проблемами обслуживания баз данных (БД). Например, прикладная программа может обратиться к системе управления базой данных (СУБД) с запросом: «Выдать фамилии тех сотрудников 42-го отдела, зарплата которых превышает зарплату их непосредственных начальников и которым до пенсии осталось работать менее 10 лет». Наличие семантического разрыва выявляет хорошо заметная разница между такой содержательной формой запроса и фактическими командами интерфейса ввода-вывода: «Блок головок дисковода с адресом 190 переместить на цилиндр 47» и «Прочитать четвертый сегмент на седьмой дорожке магнитного диска».

В настоящее время подобный семантический разрыв компенсируется введением нескольких уровней программного обеспечения (уровней интерпретации), на каждом из которых выполняется преобразование (интерпретация) запроса. Так, интерпретирующая программа СУБД преобразует исходный запрос в группу операций над конкретными наборами данных (файлами). Программа управления файлами преобразует эти операции в команды супервизора ввода-вывода. Последний формирует команды поиска, чтения и записи для исполнения модулями обмена, предназначенными для работы с определенными устройствами. Наконец, планировщик ввода-вывода для конкретных устройств инициирует запрошенные операции обмена и обрабатывает возникающие прерывания. Система такого многоуровневого преобразования запросов является весьма неэффективным средством сокращения семантического разрыва по следующим трем причинам. Во-первых, отсутствует параллелизм в работе: команды каждого уровня интерпретации, как

правило, выполняются последовательно одним процессором. Вторых, правила доступа к информации БД (например, на магнитных дисках) обычно накладывают ограничения на длину одновременно обрабатываемых данных: она соответствует длине слова или записи. В-третьих, возможности обработки данных ограничены средствами процессора, выполняющего программу данного уровня интерпретации. Как следствие обработка сравнительно несложных запросов к БД, подобных приведенным выше в качестве примера, может вызвать пересылку из внешних запоминающих устройств (внешних ЗУ) в оперативную память ЭВМ больших объемов данных, исчисляемых миллионами битов.

Очевидным решением описанной проблемы представляется использование машины такой архитектуры, которая обеспечивает параллельное выполнение операций над информацией БД, одновременную обработку групп данных и возможность обработки данных без переноса их в оперативную память, т. е. непосредственно во внешних ЗУ, где они расположены.

## АДРЕСАЦИЯ ЗНАЧЕНИЯМИ

Информация, хранимая в ЗУ, характеризуется двумя показателями: адресом и значением (содержимым ячейки памяти с этим адресом). Традиционное ЗУ устроено так, что запрос на информацию предусматривает задание адреса, результатом же запроса является значение (данные), хранимое по этому адресу. Подобная схема определения местоположения значения по его адресу удобна для многих применяемых на практике систем программирования. В частности, подобная схема необходима для реализации оператора присваивания вида

$$A := A + B;$$

где А и В — символические имена адресов ячеек памяти. Выполнение этого оператора сводится к извлечению данных, расположенных по указанным адресам, их сложению и записи результата на место первого адресата.

Однако на практике возникает необходимость и в другой, обратной форме доступа к данным, когда имеющейся в расположении информацией о данных является их значение. Примером такой ситуации может служить запрос типа: «Увеличить на заданную величину все данные, текущее значение которых равно 37». Другим примером, типичным при работе с БД, является запрос: «Выдать список сотрудников, зарабатывающих более 20 000 долл. в год». Такому запросу наилучшим образом соответствует возможность обращения за информацией о данных, хранимых в ЗУ, по их конкретному значению (20 000).

При этом в качестве результата должны быть получены адреса хранимых в памяти записей, относящихся к сотрудникам, зарабатывающим более 20 000 долл.

Возникающая в подобной ситуации проблема связана с тем, что операции, подлежащие выполнению с БД, требуют доступа к данным по значению, а физический носитель информации БД — запоминающая среда — ориентирована на доступ к данным по адресу. Такое несоответствие становится причиной значительного семантического разрыва в системе. Как следствие этого на программное обеспечение СУБД в качестве первой функции возлагается обязанность сокращения указанного разрыва путем преобразования запросов с доступом по значению в запросы с доступом по адресу.

Наиболее простой способ такого преобразования сводится к последовательному просмотру всех данных. Однако такое решение эффективно только при небольших объемах данных. Для повышения эффективности в СУБД обычно создают дополнительные структуры данных, называемые *указателями доступа* или *индексами*. Индекс — это структура типа «список», позволяющая преобразовывать требуемые значения данных в адреса местоположения этих данных. Поскольку, однако, сами индексы располагаются в памяти, доступ к содержимому которой осуществляется посредством адресов, использование индексов хотя и сокращает, но полностью не устраняет необходимости последовательности поиска. Использование индексов связано также со следующими недостатками: 1) повышенной сложностью программного обеспечения; 2) необходимостью дополнительного объема памяти; 3) необходимостью заблаговременного формирования индексов (т. е. предварительного указания полей в записях БД, в которых необходимо выполнять поиск); 4) большими «накладными расходами» на операции добавления и обновления записей в БД (так, если запись состоит из 40 полей и в БД организованы индексы для 20 из них, то при каждом добавлении одиночной записи СУБД должна обновлять 20 индексов).

Эти проблемы могут быть устранины, если использовать *ассоциативную* (адресуемую по значению ее содержимого) память. При работе с запоминающей средой подобного типа вместо запроса «Каково содержимое области с адресом X?» пользуются следующим запросом: «В каких областях памяти поле Y (например, его разряды с 7-го по 35-й) содержит величины, превышающие 20 000?».

Ассоциативная память обладает следующими свойствами:

- 1) операции в памяти выполняются не над ее отдельными элементами, а относятся сразу к группе или даже ко всем элементам;

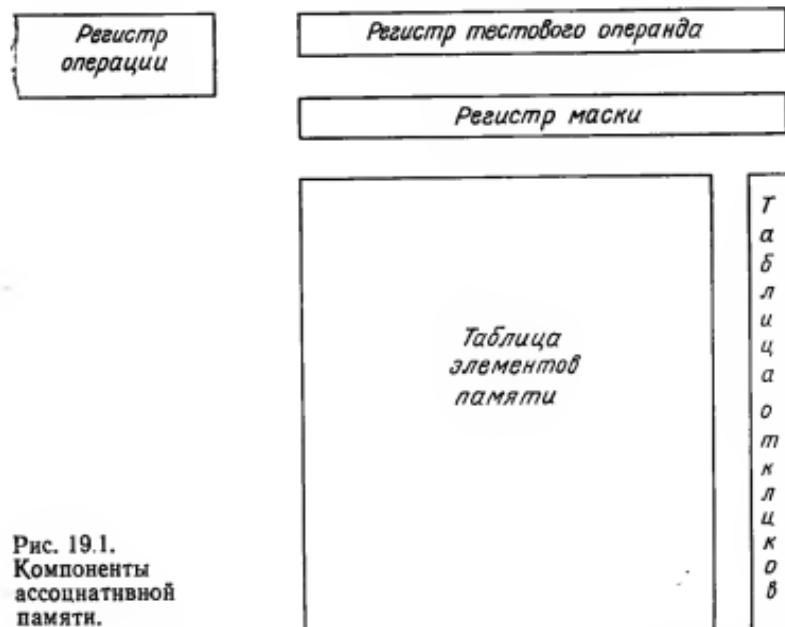


Рис. 19.1.  
Компоненты  
ассоциативной  
памяти.

2) операции в памяти выполняются одновременно над всеми ее элементами, подлежащими обработке;

3) основной операцией в памяти является поиск или сравнение;

4) время поиска (согласно п. 1—3) не зависит от числа хранимых в памяти элементов.

На рис. 19.1 изображены основные компоненты ассоциативной памяти. Таблица элементов памяти — матрица двоичных разрядов — состоит из строк — элементов памяти, каждый из которых используется, например, для размещения логической записи БД. В регистре тестового операнда размещается текущий аргумент поиска (например, число 20 000). Регистр маски определяет, на какие группы разрядов или поля элементов памяти распространяется действие аргумента поиска. Регистр операции определяет операцию, которая должна быть выполнена. Типичными операциями являются поиск значения (значений), равного тестовому; не равного ему; максимального; минимального; принадлежащего диапазону значений; ближайшего большего; ближайшего меньшего; меньшего, чем тестовое значение, и т. п.

Таблица откликов состоит из строк — элементов длиной не менее одного двоичного разряда. Каждый элемент этой таблицы используется для регистрации результата выполнения операции над содержимым элемента памяти, соответствующего

данному элементу откликов. Если содержимое текущего элемента памяти удовлетворяет критерию поиска, в разряды соответствующего элемента таблицы откликов записываются единичные биты.

Размер регистра тестового операида (операнда сравнения) и регистра маски расширен по сравнению с размером элемента памяти на величину, равную размеру элемента таблицы откликов. Это позволяет применять операции поиска как к таблице элементов памяти, так и к таблице откликов. Такая возможность используется для выполнения над содержимым таблицы элементов памяти таких операций, как пересечение или объединение множеств. Это достигается последовательностью операций, распространяемых и на содержимое таблицы откликов, хранящей «отклики» на ранее выполненные операции.

Для иллюстрации рассмотрим ассоциативную память, содержащую записи данных о сотрудниках некоторой организации. Пусть в разрядах 7—26 каждого элемента памяти содержатся сведения о заработной плате. Для решения задачи отыскания всех сотрудников, зарабатывающих более 20 000 долл., но не более 25 000 долл., достаточно использование трех команд:

```
SEARCH >20000 MASK=7..26
SEARCH (<25000 MASK=7..26) AND (=TRUE MASK=RA)
READ -TRUE MASK=RA
```

По первой команде выполняется одновременный просмотр разрядов 7—26 всех элементов памяти. Для каждого элемента, содержимое разрядов 7—26 которого окажется более 20 000, будет записана двоичная единица в соответствующий одноразрядный элемент таблицы откликов (RA). Для остальных элементов памяти в соответствующие элементы таблицы откликов записываются двоичные нули. Во второй команде заданы два условия. По этой команде требуется записать двоичные единицы в элементы таблицы откликов только для тех записей, для которых эти одноразрядные элементы уже получили двоичные единицы в результате предшествующей операции и для которых содержимое поля заработной платы (разряды 7—26 элементов памяти) меньше или равно 25 000 долл. Для остальных элементов памяти в соответствующие элементы таблицы откликов записываются двоичные нули. Третья команда осуществляет вывод записей (содержимого элементов памяти), соответствующие элементы таблицы откликов которых содержат двоичные единицы.

#### ЧАСТИЧНАЯ АССОЦИАТИВНОСТЬ

Очевидным недостатком ассоциативной организации памяти является ее высокая стоимость. Память, изображенная схематически на рис. 19.1, обеспечивает выполнение разнообразных

логических операций типа сравнения для каждого двончного разряда таблицы элементов памяти. Поэтому в ней не может быть достигнута такая плотность хранения данных, как при традиционных формах организаций памяти. Например, каждая запись в модуле ассоциативной памяти серии 10115 фирмы Signetics состоит всего лишь из 16 разрядов. Однако известны компромиссные решения, позволяющие в значительной степени сохранять достоинства ассоциативной памяти при приемлемой стоимости подобной памяти.

Отметим четыре основных типа организаций ассоциативной памяти [1]. Первый тип организаций (ему соответствует схема на рис. 19.1) — память с полным параллельным доступом. Примером конкретного технического решения организаций памяти такого типа может служить память системы PEPE фирмы Bell Laboratories [2]. Для этой системы характерна высокая степень параллелизма при обращении к данным в памяти.

Ассоциативная память второго типа организаций — это память с последовательной обработкой разрядов записей. Здесь логические операции сравнения определены только для одиночного разряда данных, а не для всех разрядов одновременно. Чтобы выполнить поиск в поле, образуемом разрядами 7—26, сначала необходимо сформировать команду для одновременного просмотра и обработки во всех записях только одного 7-го разряда, затем 8-го и т. п. Память подобного типа также обладает упомянутым в предыдущем разделе четырьмя характеристиками ассоциативной памяти с той лишь оговоркой, что операции в памяти представляются в виде циклов операций над отдельными разрядами и длительность выполнения операции пропорциональна длине анализируемого поля. Примером памяти с высокой степенью параллелизма доступа и последовательной обработкой разрядов может служить память системы STARAN [3].

Ассоциативная память третьего типа организаций — это память с последовательной обработкой слов. Здесь логические операции сравнения определены только для одиночного элемента памяти. Адресация значениями достигается при этом последовательным применением логических операций к отдельным словам — записям в памяти. Это по существу системы, в которых последовательный поиск реализуется аппаратно. Примером может служить выполнение команды SEARCH в системе SWARD. Однако, поскольку при последовательной обработке слов некоторые из упоминавшихся выше свойств ассоциативной памяти (одновременность операций и независимость длительности поиска от числа элементов в памяти) отсутствуют, такие системы не могут в полной мере считаться системами с ассоциативной памятью.

Ассоциативная память четвертого типа организации — это так называемая частично ассоциативная память — *память, организованная блоками*. Все элементы памяти формируются в  $n$  групп. Каждая группа элементов обрабатывается параллельно со всеми другими группами элементов путем последовательного перехода от одного элемента группы к другому. Следовательно, требуются средства, позволяющие последовательно выбирать элементы из группы для применения к ним заданных логических операций. Поскольку элементы в группах выбираются последовательно, оказывается возможным ограничиться использованием устройств памяти последовательного доступа вместо более дорогостоящих устройств прямого доступа.

На рис. 19.2 показан наиболее распространенный вариант реализации памяти, организованной блоками. Каждый блок представляет собой замкнутую цепочку элементов памяти (кольцо), которая в частном случае может быть построена на регистрах сдвига. Когда элемент памяти обрабатывается механизмом чтения — записи, над его содержимым выполняются логические операции ассоциативной обработки (обозначаемые на рис. 19.2 как AL).

Организация ассоциативной памяти блоками является компромиссным решением проблемы одновременности выполнения операций над содержимым всех элементов памяти (согласно второму свойству ассоциативной памяти). При этом происходит одновременное выполнение операций не над содержимым всех элементов памяти, а над содержимым группы элементов, взятых по одному из каждого блока, причем в пределах каждого блока элементы памяти обрабатываются последовательно. Привлекательной чертой такого способа организации памяти является то, что он предоставляет широкие возможности для модификации структуры подобной памяти, исходя из требований ее стоимости, быстродействия, необходимых характеристик хранения данных и выполнения логических операций. Одним предельным вариантом организации подобной памяти является структура, каждый блок которой состоит только из одного элемента памяти (т. е. это вариант полностью параллельного доступа к данным, соответствующий рис. 19.1). Другой предель-

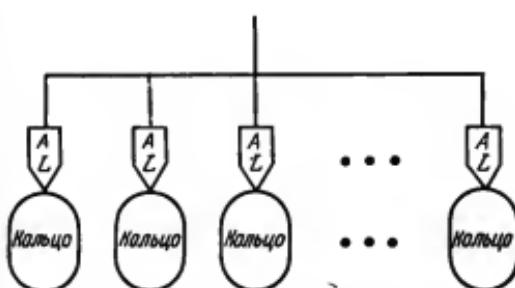


Рис. 19.2. Память, организованная блоками.

ный вариант построения подобной памяти предполагает наличие единственного блока элементов памяти — вариант последовательной обработки слов.

Скорость обработки в памяти, организованной блоками, может достигать значительной величины. Далеко не самая мощная система может состоять из 1000 блоков, в каждом из которых находится до 200 записей. Допустим, что данные перемещаются со скоростью (частотой) 100 циклов (оборотов кольца элементов памяти) в секунду. Пусть также в схемах ассоциативной логики предусмотрена возможность одновременной обработки трех полей записи (т. е. обработки запросов типа: «Выявить всех налогоплательщиков 1946 года рождения, не состоявших в браке и зарабатывающих менее 15 000 долл.»). Эта операция может быть выполнена над всем множеством данных, включающим 200 000 записей, за 10 мкс (один оборот кольца элементов памяти). Можно подсчитать среднюю частоту выполнения операций сравнения. В данном случае она оказывается равной 60 млн. операций сравнения в секунду<sup>1)</sup>. Это на много порядков выше соответствующих показателей для памяти с традиционной организацией.

Циклический просмотр данных может быть выполнен разными способами. Во-первых, в качестве памяти можно воспользоваться вращающимися магнитными дисками. При этом логические схемы ассоциативной обработки подключаются к головкам считывания и записи. Во-вторых, можно использовать регистры сдвига на базе элементов с зарядовой связью и элементов памяти на ЦМД (цилиндрических магнитных доменах), именуемой также *пузырьковой памятью*.

В зависимости от особенностей выбранной технологической базы для памяти с блочной организацией и, в частности, от быстродействия перемещения данных возможны различные способы организации передвижения данных внутри блока (по кольцу элементов памяти). Пусть записи состоят из последовательностей символов, а символы представляются последовательностью значений восьми двоичных разрядов. Наиболее типичную форму представления данных можно охарактеризовать как последовательную по разрядам и по словам. В этом случае каждая запись располагается целиком в одном кольце. Другая форма представления является последовательной по разрядам и параллельной по словам. В этом случае данные каждой записи распределяются между  $n$  кольцами, где  $n$  — число символов в

<sup>1)</sup> Единицей подсчета является операция сравнения полей записей. Для более элементарной операции сравнения одиночных двоичных разрядов значение частоты еще выше. — Прим. перев.

записи; каждый символ располагается в единственном своем кольце.

Существует также такая форма представления данных в памяти с блочной организацией, как параллельное представление по разрядам и последовательное по словам при условии объединения колец в группы по восемь. Каждая запись и каждый символ в любой записи представляются данными восьми колец: первые разряды в группе колец дают компоненты представления первого символа первой записи, вторые разряды образуют второй символ первой записи и т. д. Наконец, возможно следующее представление данных: параллельное как по разрядам, так и по словам. Число используемых при этом колец равно 8п. При этом первые разряды, считываемые со всех 8п колец относятся к первой записи, вторые — ко второй и т. д.

Отметим, что в частично-ассоциативной памяти, построенной на основе памяти на ЦМД, предпочтительнее не пользоваться представлением данных, последовательным как по разрядам, так и по словам, поскольку для этой технологии характерны низкие скорости обмена.

### ПРИМЕНЕНИЕ АССОЦИАТИВНОЙ ПАМЯТИ ПРИ ПОСТРОЕНИИ БАЗ ДАННЫХ

Значительные достоинства ассоциативной памяти проявляются при построении баз данных (см. гл. 20). Наиболее очевидное из них — высокая скорость поиска и извлечения требуемых данных. Выше отмечались причины, по которым типичные схемы последовательного поиска оказываются неудовлетворительными для поддержания работы больших БД, функционирующих в режиме реального масштаба времени. Хорошим примером большой БД с жесткими требованиями к ее производительности является база данных авиалиний. Из обзора функциональных задач аванслужбы [4] можно заключить, что основной объем операций по обработке информации приходится на следующие службы:

- система резервирования билетов;
- система составления тарифной сетки за проезд;
- система учета перевозимого веса;
- планирование занятости летного состава;
- система резервирования мест в гостиницах;
- регистрация пассажиров;
- служба связи;
- планирование рейсов;
- техническое обслуживание самолетов;
- регистрация багажа.

Из упомянутого обзора следует, что в 1980 г. для функционирования крупной авиакомпании характерен большой поток

запросов, поступающих с терминалов в СУБД. Он достигает интенсивности 175—200 запрос/с. База данных имеет объем, равный ~4 млрд. байт. При обработке 175—200 запрос/с выполняется 2400—2800 обменов с файлами БД при среднем объеме информации за один обмен, составляющем 1000 байт. Средняя периодичность обращений составляет 400 мкс. Это намного превышает возможности традиционных СУБД, работающих с большими массивами данных. Таким образом, очевидны необходимость перехода авиаслужб в высокой степени проблемно-ориентированное программное обеспечение и невозможность эффективной эксплуатации ими СУБД общего назначения.

В гл. 20 и 21 рассматриваются три экспериментальные СУБД, в которых применяются частично-ассоциативные структуры памяти с адресацией значениями. Все эти системы ориентированы на работу со структурами однородных данных. Конечно, возможности этих систем не отвечают требованиям всего разнообразия задач обработки данных. В общем случае система должна быть пригодна к обработке неформатизованных данных различной структуры. С такой ситуацией мы сталкиваемся при обработке запросов типа: «Указать названия всех статей из 60 последних выпусков *Communications of the ACM*, в которых рассматриваются вопросы организации и управления ассоциативной памятью в контексте ее применения в БД» или «Выдать сведения по всем делам федерального суда, связанным с юридическими нарушениями контрактов по использованию вычислительных ресурсов». Для решения подобных задач требуются способы организации памяти, отличные от рассматриваемых в двух следующих главах [5—8].

## ЛИТЕРАТУРА

1. Yau S. S., Fung H. S., Associative Processor Architecture — A Survey, *Computing Surveys*, 9(1), 3—27 (1977).
2. Evensen A. J., Troy J. L., Introduction to the Architecture of a 288-element PEPE, *Proceedings of the 1973 Sagamore Computer Conference on Parallel Processing*, New York, Springer-Verlag, 1973, pp. 162—169.
3. Batcher K. E., STARAN Parallel Processor System Hardware, *Proceedings of the 1974 NCC*, Montvale, NJ, AFIPS, 1974, pp. 405—410.
4. Freeman H. A. et al., Data Base Computer Research, TMA-00789, Sperry-Univac, 1979.
5. Hollaar L. A., Text Retrieval Computers, *Computer*, 12(3), 40—50 (1979).
6. Hollaar L. A., Rotating Memory Processors for the Matching of Complex Textual Patterns, *Proceedings of the Fifth Annual Symposium on Computer Architecture*, New York, ACM, 1978, 39—43.
7. Ahuja S. R., Roberts C. S., An Associative/Parallel Processor for Partial Match Retrieval Using Superimposed Codes, *Proceeding of the Seventh Annual Symposium on Computer Architecture*, New York, ACM, 1980, pp. 218—227.
8. Roberts D. C., A Specialized Computer Architecture for Text Retrieval, *Proceedings of the Fourth Workshop on Computer Architecture for Non-Numeric Processing*, New York, ACM, 1978, pp. 51—59.

---

## ГЛАВА 20

# РЕЛЯЦИОННЫЙ АССОЦИАТИВНЫЙ ПРОЦЕССОР

Хорошим примером использования принципа частичной ассоциативности при создании баз данных является реляционный ассоциативный процессор RAP (relational associative processor), спроектированный в Университете г. Торонто (Канада) в соответствии со спецификой реляционной структуры базы данных. Процессор использовался в качестве специализированной машины базы данных [1—11].

Поскольку RAP предназначался для проведения различных исследований, была запланирована разработка трех вариантов этого процессора: RAP.1 — к 1975 г., RAP.2 — к 1977 г., RAP.3 находился в стадии разработки во время написания этой книги. Рассмотрение проведем на примере процессора RAP.2 и, кроме того, укажем отличительные особенности процессоров RAP.1 и RAP.3.

Общая структура процессора RAP показана на рис. 20.1. Предполагается, что процессор RAP работает совместно с ЭВМ общего назначения, выполняющей роль так называемой ведущей машины. Последняя осуществляет связь с процессором RAP, посылая ему RAP-программу. Программа состоит из последовательности команд RAP, описывающих простые или сложные запросы к БД. Входящий в состав RAP контроллер представляет собой процессор, декодирующий команды RAP-программы, управляющий логическими схемами ассоциативного доступа к данным и обеспечивающий взаимосвязь с ведущей машиной. Блок статистической обработки, сопряженный с контроллером, используется при выполнении числовых операций над содержимым БД.

Поскольку процессор RAP ориентирован на так называемые реляционные принципы организации БД, перечислим основные понятия, имеющие отношение к указанным принципам.

В традиционном понимании БД — это совокупность логических файлов. Последние представляются в виде наборов запи-

сей, каждая из которых является совокупностью полей. Записи в логическом файле однородны: они содержат одинаковое число полей, причем п-е поле некоторой записи служит для хранения данных того же назначения и тех же характеристик, что и п-е поле любой другой записи этого логического файла.

В терминологии реляционных БД логический файл определяет *отношение*, запись называется *кортежем*, а совокупность возможных значений содержимого заданного поля является *доменом*. Имя домена эквивалентно имени поля, его номеру и по-

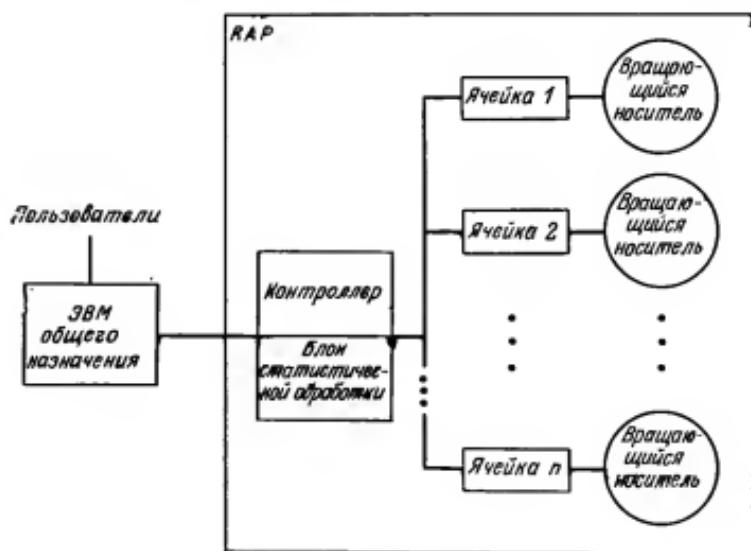


Рис. 20.1. Структура процессора RAP.

зии в записи. (Ниже термины «домен» и «поле» будут использоваться как равнозначные.) Другими словами, отношение рассматривается как двумерная таблица, содержимое которой определяет набор однотипных данных. Имя таблицы является именем отношения, каждая строка — кортежем, столбцы — доменами. Реляционная база данных представляет собой множество отношений, связанных между собой общими доменами.

В соответствии с реляционными принципами организационной БД описаннию подлежат не только логическая структура БД, но и набор основных операций над данными. Так, операции выбора позволяют в определенном отношении находить кортежи, удовлетворяющие группе задаваемых условий. Операция соединения служит для отбора кортежей из заданного отношения по критерию поиска, заданному для другого отношения; связь здесь устанавливается по общему для обоих отношений имени

домена. На подмножествах отношения определим также набор таких традиционных операций, как объединение, пересечение, дополнение и вычитание. Операция проектирования позволяет из заданного отношения выбрать подмножество доменов, исключающее дублирование их значений. Операция со свободной переменной позволяет отбирать кортежи по значениям доменов других кортежей. На доменах обычно задаются также операции модификации (обновление значений содержимого полей) и арифметические операции.

### ФУНКЦИОНАЛЬНАЯ ЯЧЕЙКА ПРОЦЕССОРА RAP

Система RAP может состоять из сотен отдельных функционально независимых ячеек, каждая из которых способна выполнять операции над содержимым базы данных ассоциативно и одновременно с другими подобными ячейками. Структура такой ячейки показана на рис. 20.2. В процессоре RAP.1 каждая ячейка — это сочетание логических схем обработки и запоминающего устройства вращающегося типа в виде дорожки диска с

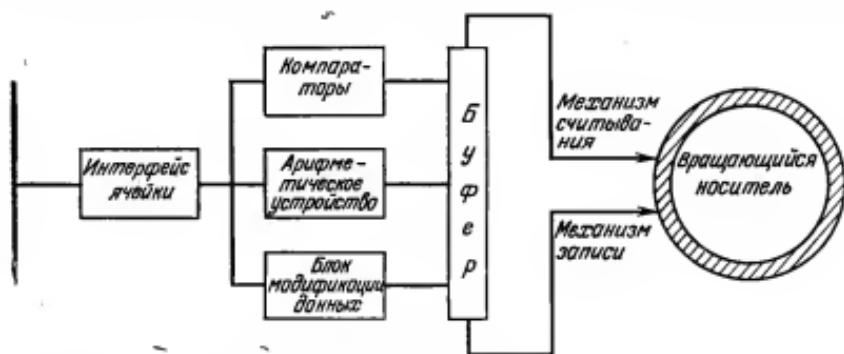


Рис. 20.2. Структура ячейки процессора RAP.

фиксированной над ней головкой. В процессоре RAP.2 вместо описанного запоминающего устройства в ячейке используется прибор с зарядовой связью, физически реализованный в виде регистра сдвига последовательного типа длиной 1 мли. бит. В процессоре RAP.3 в качестве подобной памяти для ячеек применяются регистры сдвига, построенные на элементах памяти на ЦМД, а логические схемы обработки реализуются в основном на микропроцессорах.

В каждой ячейке имеется регистр, содержащий имя хранимого отношения. Имя может храниться в закодированной форме. (В процессоре RAP.1 имя отношения хранилось в нача-

ле цепочки записей в запоминающем устройстве вращающегося типа.) Ячейка функционирует следующим образом. При прохождении кортежем считывающего механизма содержимое кортежа считывается в буфер. После этого логические схемы ячейки имеют возможность анализировать домены считанного кортежа, и, если необходимо, изменять их содержимое. Если имела место модификация, то с помощью механизма записи выполняется обновление соответствующей записи на вращающемся носителе.

Каждая ячейка процессора RAP.2 содержит три компаратора, благодаря которым возможна одновременная проверка содержимого трех доменов кортежа, находящегося в буфере (например, проверка на равенство некоторому значению, неравенство или превышение его). При этом допускается одновременное выполнение конъюнкций (И) или дизъюнкций (ИЛИ), но комбинации этих операций запрещены.

В каждой ячейке имеются также регистры для хранения значений доменов, к которым могут адресоваться RAP-программы.

## ПРЕДСТАВЛЕНИЕ ДАННЫХ

Физическое представление данных в процессоре RAP адекватно реляционной структуре БД. Взаимосвязь данных описывается отношениями, кортежами и доменами. Отношение может распространяться на группу ячеек RAP, но каждая из них может содержать кортежи только одного и того же отношения. Физические форматы данных в запоминающем устройстве вращающегося типа ячейки показаны на рис. 20.3. Кодируемая определенным образом метка обозначает начало данных (соответствует маркеру дорожки на магнитном диске). Первый блок данных состоит из имен доменов данного отношения. Имена, как правило, хранятся в виде кодов. Кодирование символьических имен осуществляют программы СУБД, выполняемые ведущей машиной.

Остальные блоки состоят из кортежей данного отношения. Первый двоичный разряд (DF) используется для размещения флага удаления. Единица, записанная в этот разряд, означает, что дальнейшее хранение кортежа не требуется; занимаемое этим кортежем место (теперь освободившееся) будет позднее учтено, как не занятое, программой «сбора мусора» данной ячейки процессора RAP. Содержимое маркерных разрядов соответствует содержимому таблицы откликов, описываемой в гл. 19. Кортеж называют Т-маркированным (где Т — любая комбинация четырех битов), если в соответствующих данному Т маркерных разрядах находятся единицы. Кортеж считается Т-немаркированным, если в указанных разрядах содержатся

## Память ячейки

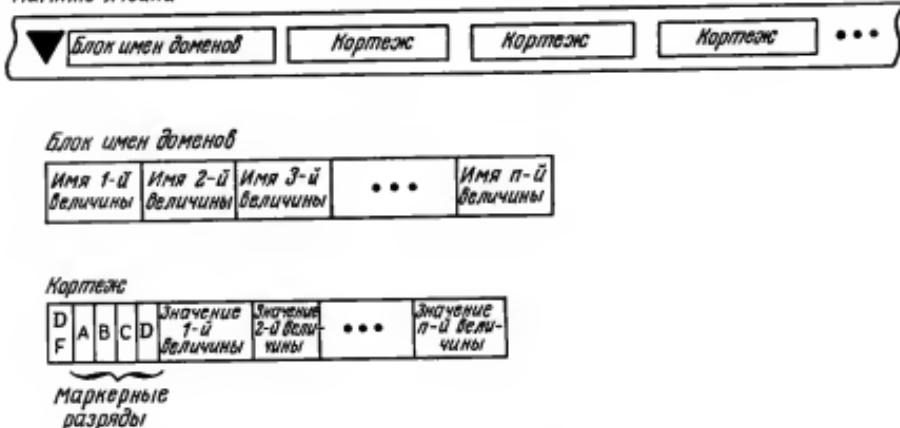


Рис. 20.3. Форматы данных для процессора RAP.

нули. Посредством команд системы RAP можно записывать двоичные единицы или нули в маркерные разряды либо проверять содержимое последних (сравнивая с тестовым значением).

Остальные поля кортежа содержат значения доменов. Их порядок должен совпадать с порядком перечисления имен в блоке имен доменов. Каждая хранимая величина может занимать 8, 16 или 32 разряда. В двух разрядах, предшествующих подобной величине, указывается длина занимаемого ею поля. Величины, имеющие числовые значения, хранятся в виде дополнительного кода. Кортеж может содержать не более 255 доменов. Перечисленные характеристики и, в частности, предельный размер отдельного поля (домена), являются сильными ограничениями для реальной БД. Однако не следует забывать, что RAP — это процессор, предназначенный для изучения данной проблемы. Кроме того, требуемое кодирование значений доменов может быть выполнено ведущей машиной.

Перед помещением кортежей в ячейку RAP память этой ячейки должна быть форматирована. Для этого необходимо, во-первых, в регистр отношения данной ячейки загрузить имя отношения, которому принадлежат данные кортежи; во-вторых, записать все блоки имен доменов, за которыми следуют блоки «пустых» кортежей, пока память не будет исчерпана. При этом выполняется также запись кода 11 в первые два разряда длины первого домена первого кортежа. Этот код символично обозначают как TKE (logical track end — конец логической записи). Система воспринимает TKE как указание на то, что данный кортеж и все следующие за ним не содержат записей и, следовательно, относятся к свободной части памяти. При последующем добавлении в данную ячейку кортежа последний будет

записываться на пустое место, ранее помеченное кодом ТКЕ. При этом код ТКЕ будет перенесен в следующую по порядку запись свободной части памяти. Если кортеж удален, то программа «сбора» мусора перемещает все имеющиеся кортежи в начале дорожки так, что все свободное место концентрируется в ее конце. При добавлении нового кортежа система проверяет все ячейки процессора RAP, соответствующие данному отношению, на наличие свободного места. Если свободного места не находится, то выделяется и форматизуется еще одна ячейка системы RAP.

Можно заметить, что в данной реализации системы выполнение сбора мусора ничем не оправдано. Поскольку вращающиеся носители заранее размечены под возможные кортежи, а выполнение команды системы RAP требует одного или нескольких оборотов носителя, перемещение заполненных кортежей в начало дорожки очевидных преимуществ не дает.

#### ФОРМАТ КОМАНД

Для процессора RAP разработан набор команд, из которых могут составляться программы. С их помощью выполняются операции над БД, имеющей реляционную структуру. RAP-программу можно рассматривать как «канальную программу» системы 370, за исключением того, что семантика команд RAP в большей мере согласована с принципами организации БД. В дальнейшем эти команды будут представлены не своими машинными кодами, а символическими средствами языка ассемблера. Помимо этого, в приводимых ниже примерах не будут соблюдаться правила, ограничивающие длину используемых символьических имен (например, допустимым будет считаться содержимое домена типа „John Smith“). Будем полагать, что либо ведущая машина допускает кодирование подобных имен, либо ограничение их длины — проблема конкретной реализации системы.

Команды RAP имеют следующий обобщенный формат:

**⟨метка⟩ ⟨КОП⟩ ⟨маркеры⟩ [⟨объект⟩: ⟨условие⟩] [⟨параметр⟩]**

Символическое имя в поле **⟨метка⟩** не является обязательным. Оно определяет адрес, используемый в командах перехода. **⟨КОП⟩** — код операции. Поле **⟨маркеры⟩** служит для задания группы маркерных разрядов, которые при выполнении команды будут установлены в 1 или 0. Например, содержимое этого поля MARK(A) означает требование установить в 1 содержимое маркерного разряда А во всех кортежах, удовлетворяющих заданному условию. Если же содержимым поля **⟨маркеры⟩** является RESET(AB), то сбросу подлежат раз-

ряды А и В во всех кортежах, удовлетворяющих условию.

Поле *«объект»* определяет имя отношения, над которым выполняется данная операция. В ряде случаев в нем же указываются и имена доменов в этом отношении. Эта часть команды имеет следующий формат:

*<Rn>* или *<Rn> (<D1>, <D2>, ... <Dn>)*, где *<Rn>* — имя отношения, а *<D1>, <D2>, ... <Dn>* — имена доменов.

Поле *«условие»* служит для записи логического выражения, используемого в качестве критерия отбора подмножества кортежей, к которым должна применяться операция ассоциативной адресации. Логическое выражение может иметь одну из следующих трех форм: 1) вырожденную форму, эквивалентную отсутствию условия (предполагается, что адресуемы все кортежи данного отношения); 2) форму конъюнкции элементарных условий; 3) форму дизъюнкции элементарных условий. Элементарное условие в логическом выражении должно иметь следующий вид:

- 1) *<имя><опер. сравнения><операнд>*
- 2) *<Rn>.MKED(<T>)*
- 3) *<Rn>.UNMKED(<T>)*

В поле *«имя»* задаются имена отношения и домена в форме *(<Rn>.<Dn>)*. В поле *«опер. сравнения»* указывается символ операции сравнения (равно, не равно, меньше, меньше или равно, больше, больше или равно). В поле операнда может указываться регистр RAP, литерал или символьическое имя переменной (адрес) в ведущей машине. При дальнейшем изложении материала нечисловые литералы будут заключаться в кавычки, а символьические имена переменных — в круглые скобки. В поле *<T>* указывается конкретное содержимое маркерных разрядов. Оно может кодироваться одним из следующих способов: A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD и ABCD.

Приведем пример задания условия:

*EMP.SALARY>5000) & (EMP.MKED(A))*

Согласно этому условию, для ассоциативной обработки выделяются кортежи отношения EMP со значением заработка, превышающим 5000, в которых содержимым маркерного разряда А является 1<sup>1)</sup>.

<sup>1)</sup> Здесь и далее имена, используемые в полях команд, являются сокращениями английских слов — названий, принятых в рассматриваемой задаче для отношений и доменов. Для удобства читателя приведем сокращения, используемые в данной главе: EMP — сотрудник, DEPT — отдел, SAL — зарплата, VOL — количество, MGR — управляющий, PARTNUM — шифр товарного изделия. Употребляются также полные имена: SALARY — зарплата, NAME — фамилия, ITEM — наименование изделия. — Прим. перев.

Аппаратная реализация накладывает ограничения на форму записи составных условий и на допустимое количество используемых при этом элементарных условий. В частности, в процессоре RAP.2 составное условие формируется не более чем из трех простых.

Последнее поле *«параметр»* в формате команд RAP используется только в командах некоторых типов и будет рассмотрено позднее, когда возникнет в этом необходимость.

### НАБОР КОМАНД ПРОЦЕССОРА RAP

Процессор RAP располагает обширным набором машинных команд, с помощью которых, используя имеющееся в ведущей машине программное обеспечение СУБД, можно создавать программы управления БД. Поскольку функции некоторых из них весьма сложны, не все эти команды будут описаны полностью. Обсуждение будет ограничено той степенью детальности, которая необходима для понимания их функционального назначения. Операции, выполняемые командами, будут рассматриваться на примере БД, изображенной на рис. 20.4. Заметим, что у обоих рассматриваемых отношений (EMP и ITEM) этой БД имеется общий домен DEPT.

Время выполнения команд удобно исчислять количеством циклов обращения носителей запоминающего устройства врачающегося типа. В случае когда это возможно, такая характеристика приводится ниже. Иногда время выполнения зависит от особенностей аппаратной реализации системы или от самих данных.

Перейдем к рассмотрению отдельных команд. Команда SELECT<маркеры>[<Rn> : <условие>] переводит в нужное состояние указанные разряды маркеров всех кортежей отношения Rn, удовлетворяющих заданному условию. В эти разряды записываются двоичные единицы или нули. Время выполнения равно одному обороту врачающегося носителя (диска).

Например, если имеется файл сотрудников (отношение EMP, то по команде

SELECT MARK(AB) [EMP : EMP.DEPT=202]

двоичные единицы записываются в разряды А и В кортежей сотрудников, работающих в отделе 202. Команда

READ-ALL<маркеры>[<Rn>(<D1>, ... ) : <условие>][<адрес>]

пересылает удовлетворяющие заданному условию кортежи в указанную область ведущей машины. Пересылаться могут или все домены выбираемых по условию кортежей или их подмножество. Если в команде присутствует поле «маркеры», в соответствующие разряды записываются двоичные единицы или нули. Время выполнения зависит от количества вы-

Рис. 20.4. Пример базы данных.

бираемых кортежей и числа каналов ввода-вывода, связывающих процессор RAP с ведущей машиной. В частности, по команде

READ-ALL[EMP:MGR='SMITH'] [BUF]

из поля EMP в поле BUF пересыпаются данные о всех сотрудниках отдела, возглавляемого Смитом (Smith). С помощью команды

READ-ALL[EMP(NAME, SAL) : EMP.MKED(A)][BUF]

производится запись в поле BUF фамилии и заработной платы (значения доменов NAME и SAL) тех сотрудников, в записях которых в данный момент отмечен маркер А. Следующая команда

**READ[<n>]<маркеры>[<Rn>(<D1>, ...): <условие>][<адрес>]**

эквивалентна команде READ-ALL, за тем исключением, что в память ведущей машины пересылаются не все кортежи, удовлетворяющие заданному условию, а только первые  $p$  из них. Заметим, однако, что понятие «первых записей» кортежей в системах ассоциативной памяти по существу не определено. Все зависит от конкретной реализации процессора RAP, а так-

же от того, как именно расположились записи на вращающихся носителях в отдельных ячейках RAP. Команда

**READ-REG[<список регистров>][<адрес>]**

копирует содержимое указанных регистров RAP в память ведущей машины по заданному адресу. (В процессоре RAP.2 имеется три адресуемых регистра.) Время выполнения этой команды меньше одного оборота вращающегося носителя. Команда

**CROSS-SELECT<R1 маркеры>[<R1>:<D1><опер. сравнения> <R2>.<D2>] [<R2>.MKED(<T>)]**

присваивает значения содержимому маркерных разрядов кортежей отношения R1 в соответствии с результатом сравнения домена D1 отношения R1 и домена D2 отношения R2. При этом используется только Т-маркированное содержимое домена D2 во всех кортежах R2. Конечно, содержимое доменов D1 и D2 должно допускать сопоставление. Возможна ситуация, при которой для каждого кортежа R2 имеется более одного кортежа R1, удовлетворяющего условию сравнения.

Команда выполняется за  $1 + NT2/K + N$  оборота вращающихся носителей, где  $NT2$  — количество Т-маркированных кортежей в R2,  $N$  — число ячеек RAP, содержащих Т-маркированные кортежи в R2, а  $K$  — величина, зависящая от конкретной реализации и равная количеству данных отношения R2, которые могут одновременно пересыпаться в ячейки RAP, содержащие данные отношения R1. Рассмотрим последовательность из двух команд:

```
SELECT MARK(A) [ITEM:ITEM.VOL>100]
CROSS-SELECT MARK(A) [EMP:EMP.DEPT=ITEM.DEPT]
[ITEM.MKED(A)]
```

В результате выполнения второй команды будут отмечены записи о служащих, работающих в отделах, где продано более 100 единиц продукции.

**CROSS-COND-SELECT <R1 маркеры><врем. маркеры> [<R1>:<D1><опер. сравнения> <R2>.<D2>] [<R2>.MKED(<T>)]**

Эта команда близка к команде CROSS-SELECT. Указанные маркеры записей отношения R1 устанавливаются в 0, если условие не выполняется. Параметр <врем. маркеры> указывает один или несколько маркерных разрядов, которыми можно пользоваться во время выполнения данной команды. Такую команду применяют для получения пересечений (логическое И) двух подмножеств одного и того же отношения. Обычно этой команде предшествует команда CROSS-SELECT. Время выполнения команды CROSS-COND-SELECT превышает время вы-

полнения команды CROSS-SELECT на длительность одного цикла.

#### Команда

**GET-FIRST-MARK***<маркеры>[<Rn>(<D1>, ...):  
<Da><опер.сравнения><Db>] [MKED(<T>)]*

используется для выполнения операции проектирования в пределах одного и того же отношения. Значение домена *<Db>* из-которого Т-маркированного кортежа последовательно сравнивается со значением домена *<Da>* всех кортежей данного отношения. Значения доменов *<Da>* и *<Db>* могут указывать на один и тот же или на разные домены отношения. В последнем случае в них должны храниться сопоставимые величины. При выполнении команды системой производятся следующие действия: 1) маркерам всех кортежей, удовлетворяющих заданному условию, присваиваются значения в соответствии с содержимым поля *<маркеры>* данной команды; 2) сбрасывается Т-маркер кортежа, относительно которого выполнялось сравнение и 3) значения указанных доменов (*<D1>, ...*) этого кортежа записываются на хранение в регистры RAP. (В RAP.2 на хранение можно записывать содержимое не более трех полей; для этого используются регистры 1—3.)

Пример использования этой команды будет приведен ниже. Время ее выполнения в среднем равно полутора периодам вращения носителя. Команда

**GET-FIRST**[<Rn>(<D1>, ...)][MKED(<T>)]

выполняется подобно предыдущей. В ией, однако, отсутствует проверка условия и не маркируются соответствующие кортежи, т. е. исключен первый шаг из действий, реализуемых по команде GET-FIRST-MARK. Т-маркер того кортежа, в котором он был установлен, сбрасывается, а значения указанных доменов этого кортежа загружаются в регистры RAP. По команде

**SAVE(<n>)[<Rn>(<D1>, ...):<условие>] [<список регистров>]**

в указанные регистры загружаются значения перечисленных доменов первых *n* записей, удовлетворяющих заданному в команде условию. Если используется необязательный операнд *<маркеры>*, то кортежи, удовлетворяющие указанному условию, маркируются. Команда выполняется за время, равное одному циклу.

Команду GET-FIRST можно рассматривать как частный случай более общей команды SAVE. Здесь приводятся обе команды вследствие некоторых несоответствий нескольких известных описаний списка команд RAP.2.

Рассмотрим пример. В результате выполнения команды  
**SAVE(1)[ITEM(PARTNUM):ITEM.VOL < (MIN)] [REG1]**

система выделит из товарного перечня изделие ITEM, количество единиц которого (значение домена VOL) меньше значения переменной MIN, и поместит шифр этого изделия в регистр 1. Следующие команды

```
ADD      <маркеры>[<Rn>(<Dn>):<условие>][<операнд>]
SUB      <маркеры>[<Rn>(<Dn>):<условие>][<операнд>]
REPLACE <маркеры>[<Rn>(<Dn>):<условие>][<операнд>]
```

имеют одинаковый формат. Выполняются они похожим образом. А именно, требуемая операция<sup>1)</sup> осуществляется над доменами  $\langle D_n \rangle$  всех кортежей, удовлетворяющих заданному условию. Если содержимое поля <маркеры> определено, то выполняется маркирование указанных кортежей по мере их просмотра системой. Это используется для восстановления работоспособности системы: в таком случае выполнение операции может быть возобновлено над множеством неотмеченных записей. Время выполнения равно одному периоду обращения носителя.

Рассмотрим пример. В результате выполнения команды

**SUB[ITEM(VOL):ITEM.DEPT=410&ITEM.VOL>100][1]**  
 1 вычитается из содержимого домена (VOL) количества единиц различных видов изделий, выпускаемых отделом 410, которые имеют в этом домене значение более 100. Команды

```
SUM      <маркеры>[<Rn>(<Dn>):<условие>][<регистр>]
MAX      <маркеры>[<Rn>(<Dn>):<условие>][<регистр>]
MIN      <маркеры>[<Rn>(<Dn>):<условие>][<регистр>]
COUNT    <маркеры>[<Rn>(<Dn>):<условие>][<регистр>]
```

задают функции, определенные на множествах. Первые три команды (вычисление суммы, максимума и минимума) вычисляют значение функции по множеству значений домена  $\langle D_n \rangle$  кортежей, удовлетворяющих заданному в команде условию. По команде COUNT подсчитывается количество таких кортежей. Вычисленное значение загружается в указанный регистр. Если задается operand <маркеры>, выделенные кортежи маркируются. Время выполнения несколько больше одного периода обращения носителя.

В частности, по команде **SUM[ITEM(VOL)][REG1]** вычисляется сумма значений в поле количества (VOL) всех кортежей из отношения ITEM. По команде

**DELETE[<Rn>:<условие>]**

<sup>1)</sup> Операция сложения с operandом, вычитания operand'a или замены на operand. — Прим. перев.

исключаются все кортежи, удовлетворяющие заданному условию, т. е. разряд флага удаления этих записей устанавливается в 1. Эта команда выполняется за один период обращения носителя. По команде

**INSERT(п) [ $\langle Rn \rangle$ :<список ячеек>] [<адрес>]**

кортежи, расположенные в ведущей машине по указанному адресу, добавляются к хранимым в базе данных кортежам указанного отношения. Если содержимое поля «список ячеек» определено, то кортежи помещаются только в указанные ячейки. Время выполнения равно одному периоду обращения носителя, если в ячейках, которые содержат  $\langle Rn \rangle$ , имеется свободное место. В противном случае система назначает дополнительную ячейку и форматизирует ее. При этом время выполнения оказывается равным двум периодам обращения носителя. По команде

**DESTROY[ $\langle Rn \rangle$ :<список ячеек>]**

из памяти удаляется все отношение, если содержимое поля «список ячеек» не определено; иначе удаляются все кортежи данного отношения только из указанных ячеек. Освободившиеся ячейки могут быть использованы для хранения данных других отношений. По команде

**CREATE[ $\langle Rn \rangle$ :<список ячеек>] [<адрес>]**

для заданного отношения форматизируются указанные ячейки. По указываемому операндом этой команды адресу содержимое поля «адрес» указывает местоположение в ведущей машине информации об именах и размерах доменов в кортежах отношения  $\langle Rn \rangle$ . По команде

**COLGRBG[<список отношений>и/или<список ячеек>]**

выполняется „сбор мусора” в указанных отношениях или ячейках. Сохраняемые записи смещаются к началу циркулирующего носителя на место записей, отмеченных флагом удаления. По команде

**SPACE-COUNT[ $\langle Rn \rangle$ :<список ячеек>] [<регистр>]**

выполняется подсчет свободных позиций данного отношения по всем или только по указанным ячейкам. Результат загружается в заданный регистр.

По команде

**BC <метка>,<условие перехода>**

осуществляется передача управления команде с указанной мет-

кой, если выполнено условие перехода. Последнее может иметь следующую форму записи:

1. <регистр><опер. сравнения><константа>
2. <регистр><опер. сравнения><регистр>
3. TEST(<Rn>:<маркер условия>)

Время выполнения этой команды менее одного периода обращения носителя. Например, по команде BC LOOP, REG2=0 управление передается команде с меткой LOOP в случае, если содержимое регистра REG2 равно нулю.

Команда BC используется для организации циклов в RAP-программах, что, в частности, необходимо для выполнения операций проектирования. Команды

INSERT-REG	[<список регистров>][<список констант>]
DECREMENT-REG	[<регистр>]
INCREMENT-REG	[<регистр>]
RADD	[<регистр>][<операнд>]
RSUB	[<регистр>][<операнд>]
RMUL	[<регистр>][<операнд>]
RDIV	[<регистр>][<операнд>]

выполняют операции записи, уменьшения, увеличения, сложения, вычитания, умножения и деления над содержимым одного или нескольких регистров. Содержимое поля «операнд» используется как указатель регистра или константы.

Команда EOQ завершает выполнение RAP-программы.

## ПРИМЕРЫ ПРОГРАММ

Для иллюстрации рассмотрим три примера RAP-программы. Первая программа исключает из отношения EMP записи о всех сотрудниках, работающих в отделе Джонса (J. Jones). Эта программа выглядит следующим образом<sup>1)</sup>:

```
SAVE [EMP(MGR):EMP.NAME='J.JONES'][REG0]
DELETE [EMP:EMP.MGR=REG0]
EOQ
```

Время выполнения программы составляет величину немного более двух периодов (плюс в среднем еще половина периода вследствие асинхронности начала выполнения программы по отношению к текущему положению врачающегося носителя).

Вторая RAP-программа предназначена для подсчета коли-

<sup>1)</sup> Программа может состоять из двух команд: DELETE [EMP:EMP.MGR='J. Jones'] и EOQ. Заметим также, что команда SAVE использована в неявленном ранее варианте: с пропуском значения п. Автор ориентируется на версию RAP, в которой в SAVE по умолчанию п=1. — Прим. перев.

чества видов товарной продукции и общего объема товаров, проданных отделом, в котором служит Смит (C. Smith):

1. SELECT	MARK(A) [EMP:EMP.NAME='C.SMITH']
2. CROSS-SELECT	MARK(A) [ITEM:DEPT=EMP.DEPT]
	[EMP:MKED(A)]
3. COUNT	[ITEM:MKED(A)][REG1]
4. SUM	[ITEM(VOL):MKED(A)][REG2]
5. READ-REG	[REG1,REG2][BUF]
6. SELECT	RESET(A)[ITEM]
7. SELECT	RESET(A)[EMP]
8. EOQ	

По команде 1 для служащего Смита маркируется кортеж EMP. По команде 2 маркируются все кортежи ITEM, относящиеся к отделу, в котором работает Смит. С помощью команд 3 и 4 подсчитывается количество видов маркированной товарной продукции и общий объем последней; результаты размещаются в регистрах 1 и 2. По команде 5 эти данные переписываются в область BUF оперативной памяти ведущей машины. Последние две команды SELECT устанавливают в нулевое положение все использованные маркерные разряды.

В третьей из рассматриваемых RAP-программ выполняются операции проектирования и условного перехода. Пусть требуется подсчитать число отделов, в которых имеются сотрудники, заработок которых превышает 30 000 долл. Воспользоваться одной лишь командой COUNT не представляется возможным, поскольку не исключены случаи, когда в каком-либо одном или нескольких отделах окажется более чем по одному сотруднику, удовлетворяющему указанному критерию отбора; нам же требуется подсчитать лишь количество отделов. Поставленная задача решается с помощью следующей RAP-программы:

1.      SELECT	MARK(AB)	[EMP:EMP.SALARY>30000]
2.      LOOP GET-FIRST-MARK	MARK(C)	[EMP:EMP.DEPT=EMP.DEPT][MKED(B)]
3.      SELECT	RESET(ABC)	[EMP:EMP.MKED(C)]
4.      BC		A.TEST([EMP:MKED(B)])
5.      COUNT		[EMP:EMP.MKED(A)][REG1]
6.      EOQ		

В целях конкретизации анализа работы данной программы предположим, что первая команда SELECT «маркирует» служащих Смита и Зиска из 14-го отдела, Хилса, Милса и Уилса — из 19-го отдела и Уайта — из 20-го отдела. При первом выполнении команды GET-FIRST-MARK будет обнаружен один из пяти упомянутых маркированных кортежей. Предположим, что первым окажется кортеж, соответствующий служащему Смиту. В результате выполнения этой команды станут C-маркированными все остальные В-маркированные кортежи того же отдела,

сотрудником которого является Смит; при этом В-маркер Смита будет сброшен. Следующая команда SELECT выполнит сброс маркеров ABC во всех С-маркированных кортежах (относящихся к служащему Зиску).

При выполнении команды BC будет обнаружено, что существуют также В-маркированные кортежи. Поэтому произойдет повторение указанной последовательности команд. Пусть на этот раз команда GET-FIRST-MARK найдет служащего Милса, выполнит С-маркирование кортежа, соответствующего служащим Хилсу и Уилсу, и осуществит сброс В-маркера кортежа Милса. Следующая команда SELECT произведет сброс маркерных разрядов ABC кортежей, относящихся к служащим Хилса и Уилса. И опять команда BC найдет В-маркер (только в кортеже B), после чего будет повторено выполнение указанной последовательности команд. Теперь команда GET-FIRST-MARK установит в 0 В-маркер кортежа Уайта, однако больше не обнаруживаются кортежи, нуждающиеся в маркировании. Последующая команда SELECT не выявит кортежей, удовлетворяющих условию. Команда BC не обнаруживает В-маркированных кортежей. Команда COUNT вычислит требуемую величину, оказывающуюся в данном случае равной 3 (Смит, Милс и Уайт), и загрузит ее в регистр 1.

### ЭФФЕКТИВНОСТЬ ПРОЦЕССОРА RAP

Наиболее очевидным преимуществом архитектуры реляционного ассоциативного процессора перед традиционными принципами организации БД с адресацией по положению является его быстродействие, обусловленное следующими тремя факторами:

1. *Параллельное выполнение операций.* Естественно, что большие отношения (файлы) охватывают значительное число ячеек, которые допускают одновременное выполнение большинства операций поиска и модификации (обновления) информации. В традиционных же системах с обратными списками или индексами все операции выполняются последовательно. Например, для отбора данных по значениям определенного домена индекс считывается в оперативную память процессора, осуществляется последовательный поиск и затем каждая требуемая запись последовательно считывается.

2. *Обновление данных на месте.* Поскольку процессор RAP функционирует согласно принципу ассоциативной обработки, обновление данных сводится к замене соответствующего кортежа. В то же время в традиционных БД, базирующихся на принципе индексирования, изменения необходимо вносить как в сам кортеж, так и в индекс. Если для изображенной на рис. 20.4 БД сотрудников торговой фирмы индексы организованы для всех

доменов файла сотрудников, то добавление нового кортежа в этот файл должно сопровождаться изменением четырех индексов.

3. *Сокращение числа пересылок данных.* Объемы данных, пересылаемых между процессором ведущей машины и памятью (БД), в процессоре RAP на несколько порядков меньше, чем в традиционных системах. Наиболее разительно это проявляется в выполнении логических операций и при модификации данных. Если требуется в файле сотрудников увеличить длительность отпуска (значения соответствующего домена) на одну неделю, то из ведущей машины в процессор RAP должны быть переданы только две команды, а в обратном направлении — признак конца выполнения. В обычных же СУБД каждая запись (кортеж) должна быть считана в память процессора, изменена и записана на носитель. При требовании подсчета количества записей, удовлетворяющих некоторому условию, в процессор RAP пересыпается только одна команда. А в традиционных системах требуется пересылка в ведущую машину всего индекса.

С использованием методов численного моделирования [5] было проведено сравнение скорости выполнения операций в процессоре RAP и в обычных системах. В табл. 20.1 даны относительные значения скорости выполнения основных операций в процессоре RAP по сравнению с обычными системами. Колонки таблицы соответствуют случаям, когда в БД находятся соответственно 500 или 3000 кортежей, удовлетворяющих заданному критерию отбора. При расчетах предполагалось, что память системы RAP состоит из 100 ячеек по 1000 кортежей в каждой. Сравнение проведено с БД традиционной организации, имеющей память на дисках с инвертированными списками.

Таблица 20.1. Относительные значения скорости выполнения операций в системе RAP по сравнению с традиционными системами

Операция	Скорость (в отн. ед.) при количестве кортежей, удовлетворяющих критерию отбора, равном	
	500	3000
Поиск по простому условию	12	55
Поиск по сложному условию	18	45
Простая модификация	1000	5000
Сложная модификация	300	1600
Простой отбор/подсчет/выдача	35	60
Сложный отбор/подсчет/выдача	20	35
Присоединение	0,1—1	0,5—4

«Простые» операции в табл. 20.1 — операции с одним условием типа равенства. «Сложные» операции предусматривают проверку десяти простых условий. Простая операция типа отбор/подсчет/выдача может использоваться, например, при выполнении запроса: «Выдать список сотрудников, зарабатывающих больше вице-президента компании». Отношение скоростей выполнения операции присоединения зависит от того, реализуется ли в системе RAP с целью сокращения времени выполнения операции присоединения предварительная операция проектирования. Поэтому в последней строке таблицы данные приведены в виде диапазонов значений этой величины (см. упр. 20.7 и 20.8 в конце главы).

### ВЕРСИЯ RAP.3

В новой версии RAP сняты некоторые ограничения, присущие RAP.2. С целью повышения скорости обработки применены особый способ хранения данных и новый принцип организации ячейки памяти [9]. Для повышения пропускной способности используется последовательный по словам способ доступа к данным на вращающемся носителе вместо последовательного по разрядам. Кроме того, в пределах каждой ячейки организовано несколько секций, что позволяет обрабатывать несколько кортежей в ячейке одновременно.

В известной реализации RAP.3 ячейка состоит из четырех секций. Анализ и модификация содержимого кортежей на вращающемся носителе могут выполняться независимо в каждой из секций. Секции реализованы на 16-разрядных микропроцессорах серии 8086. В результате построение ячейки процессора RAP сводится не столько к проектированию логических схем, сколько к созданию программных модулей микропроцессоров 8086. Благодаря этому, во-первых, уменьшается число компонентов: несмотря на то что каждая ячейка состоит из четырех секций, общее количество модулей в системе RAP.3 составляет только третью часть количества модулей RAP.2. Во-вторых, оказывается возможным выполнять некоторые более «гибкие» операции. Синхронизация работы ячейки во времени такова, что пока в одну секцию загружается кортеж, в двух других секциях выполняется совместная обработка двух кортежей, а в четвертой секции может выполняться также операция записи кортежа на вращающийся носитель.

Процессор RAP.3 обладает лучшими эксплуатационными характеристиками. В частности, количество маркерных разрядов в кортеже увеличено до 16. В домены можно записывать символьную информацию. Домены могут быть произвольной длины. Единственным условием является следующее: общая

длина кортежа не должна превышать 1024 байт. Включены команды сравнения доменов.

В процессоре RAP.3 увеличена скорость выполнения операции объединения (CROSS-SELECT), выполнявшаяся в RAP.2 относительно медленно. Это оказалось возможным благодаря двум нововведениям. Во-первых, роль вращающейся памяти в RAP.3 выполняет память на ЦМД. Это позволяет останавливать и возобновлять процессы циклического обмена в любой момент времени и, следовательно, более эффективно выполнять команду CROSS-SELECT. Во-вторых, скорость выполнения команды CROSS-SELECT в процессоре RAP.2 ограничивалась допустимыми малыми значениями параметра k (числом доменов, значения которых из взятого за основу отношения учитывались при выполнении сравнений по CROSS-SELECT). Для процессора RAP.2 эта величина равнялась 3. В процессоре RAP.3 вместо ограничения на величину k наложено ограничение на общую длину сравниваемых данных. Эта длина не должна превышать 800 байт (например, 400 2-байтовых полей).

### ИСПОЛЬЗОВАНИЕ ПРОЦЕССОРА RAP В БОЛЬШИХ СУБД

Подсчитано, что наиболее рационально использовать процессор RAP при объеме данных  $10^8$ — $10^9$  бит. Поскольку многие БД являются хранилищем большого объема информации, была предложена модификация RAP с использованием принципов виртуальной памяти [3, 6].

В этой новой разработке все данные хранятся в основной памяти неассоциативного типа. Каждая ячейка процессора RAP располагает двумя вращающимися носителями. В любой данный момент времени один из них выполняет функции активной памяти, а другой — буфера. Как следует из рис. 20.5, оба носителя связаны с ячейкой через переключатель. В то время как один из них подсоединен к ячейке, другой подключается к процессору (контроллеру) ввода-вывода, посредством которого выполняется обмен данными с основной памятью.

Одни из вариантов функционирования системы может заключаться в следующем. Закончив формирование очередной RAP-программы, ведущая машина определяет, содержатся ли необходимые отношения в памяти процессора RAP. Если их в памяти нет, то процессор ввода-вывода выполняет пересылку этих отношений (опережающую запись «страницами») из основной памяти в ячейки процессора RAP. В это время система RAP может продолжать обработку какой-либо другой программы. Алгоритм перелистывания страниц памяти в первую очередь может выполнять поиск тех ячеек вращающихся носите-

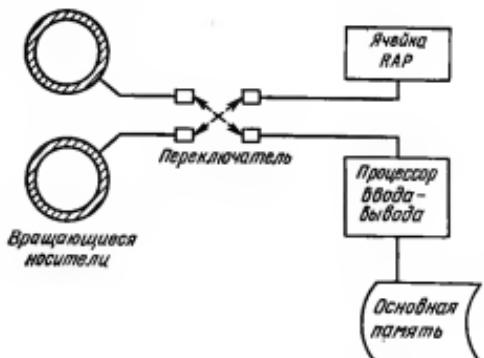


Рис. 20.5. Ячейка RAP, предназначенная для реализации перелистывания страниц в памяти.

вращающихся носителей, которые должны быть переключены на альтернативный режим работы (так называемый *сполинг*): активный носитель превращен в буферный, а буферный в активный. При использовании другого варианта функционирования управление пересылкой данных может быть возложено не на ведущую машину, а на процессор ввода-вывода системы RAP.

В основу метода перелистывания страниц положено правдоподобное, хотя и строго не проверенное предположение о том, что поток запросов к БД отличается свойством «локальности ссылок» подобно аналогичному свойству потока запроса программ к частям оперативной памяти, наблюдаемому в системах с виртуальной организацией памяти. Можно высказать следующие соображения в пользу такого интуитивного решения. По мере приближения некоторых критических моментов времени (например, запланированного времени отправления самолета) заметно возрастает интенсивность работы с определенными отношениями (в данном случае — с отношением, содержащим информацию об этом рейсе). Это вызывает эффект повышенной локальности запросов к БД. Аналогичным образом использование интерактивных языков запросов благоприятствует режиму последовательных просмотров БД (поиску подходящих данных на основе планомерной модификации запроса). В этом случае также на короткие периоды времени повышается частота обращения к отдельным группам отношений БД. Наконец, для программ пакетной обработки и для программ — генераторов отчетов характерны многократные повторные обращения к одной и той же группе отношений БД, что тоже приводит к повышенной локальности ссылок.

лей, содержимое которых не обновлялось и к которым в соответствии с текущим состоянием системной очереди RAP не запланирован доступ обрабатывающих программ. Если имеющиеся в распоряжении носители ячеек содержат только модифицированные отношения, то они сначала постраничночитываются в основную память. По завершении пересылки данных в процессор RAP пересыпаются программа и информация о конкретной паре вра-

## ДОСТОИНСТВА ПРОЦЕССОРА RAP

Выше был отмечен ряд достоинств процессора RAP и, в частности, большая скорость выполнения операций. Отметим еще некоторые важные достоинства процессора RAP и аналогичных систем ассоциативной памяти. Одно из них заключается в том, что для БД, выполненной по схеме RAP, требуется намного меньший объем памяти, чем при ее реализации по традиционной (неассоциативной) схеме. Дело в том, что в обычной системе в целях сокращения среднего времени доступа к данным требуется применение индексов или обратных списков. Для отдельного файла (отношения) со 100 полями или доменами может оказаться необходимой организация индексов для 10 или большего числа полей. Когда индексов много, требуемая для их хранения память может оказаться сравнимой с основной памятью БД или даже превысить ее.

В связи со сложностью организации системы индексов в традиционных СУБД при планировании структуры БД проявляется еще одно преимущество процессора RAP. Применительно к традиционным системам представляются практически нецелесообразными организация и ведение индексов для всех полей записей БД. Во-первых, это привело бы к очень большому расходу памяти. Во-вторых, если и создать такую систему индексов, то операции обновления существующих записей и добавления новых записей будут выполняться медленно, поскольку с поступлением каждой новой записи требуется изменение каждого индекса. Поэтому проектировщик БД стоит перед сложной проблемой выбора подмножества полей, к которым следует применить индексирование (т. е. проблемой указания полей, по которым будут выполняться операции поиска). Прогноз проектировщика может оказаться неудачным (например, вначале предполагалось, что не потребуется выполнять поиск записей в файле сотрудников по размеру заработной платы, а позднее такая необходимость возникла). В связи с использованием процессора RAP подобные проблемы не встают, поскольку адресация выполняется ассоциативным образом.

Другим достоинством процессора RAP являются большие возможности к расширению БД. Допустим, что возникла необходимость в увеличении объема БД на 50% (например, вследствие слияния корпораций) при одновременном сохранении временных характеристик доступа к данным на прежнем уровне. В СУБД со средствами индексирования время доступа пропорционально (но нелинейно) числу записей, и для обеспечения прежнего времени доступа потребовался бы процессор с более высокой скоростью выполнения операций. Однако такое решение может оказаться неприемлемым, например, вслед-

ствие уникальных характеристик имеющегося в данный момент процессора, или из-за того, что этот процессор является наиболее высокопроизводительным в своем классе, или в связи с тем, что переход к процессору с более высоким быстродействием сопровождается недопустимо резким увеличением отношения стоимость/производительность. В системах же типа RAP вычислительная мощность растет по мере наращивания памяти в процессе подключения дополнительных секций.

Из других достоинств этих систем можно отметить меньшую сложность программного обеспечения СУБД в ведущей машине и облегченный режим работы последней.

## УПРАЖНЕНИЯ

20.1. Для БД, структура которой изображена на рис. 20.4, составьте RAP-программу, в соответствии с которой производится увеличение на 100 долл. размера годового оклада (SAL) всех сотрудников отделов (DEPT) с наивысшим показателем количества изделий (VOL) по одному или нескольким видам продукции (PARTNUM). Эта операция не должна выполняться для сотрудников с  $SAL > 20\ 000$ .

20.2. Рассчитайте ожидаемое время выполнения программы, описанной в предыдущем упражнении в предположении, что период просмотра ассоциативной памяти (период оборота вращающегося носителя) равен 20 мс, число сотрудников в отделах, удовлетворяющих поставленному условию, равно 8, а  $k=5$ .

20.3. Наиболее очевидное преимущество системы RAP по сравнению с традиционными СУБД заключается в возможности параллельного выполнения операций. Назовите другие важные показатели подобных систем, определяющие их высокую производительность.

20.4. Почему относительные скорости выполнения операций процессора RAP по сравнению с традиционными системами меньше для «сложных» операций (см. табл. 20.1)?

20.5. Пусть БД, обслуживаемая системой типа RAP, представлена тремя отношениями: АДРЕСА, АБОНЕНТЫ и ТАРИФЫ. В отношение АДРЕСА входят три домена: ГОРОД, КОЛИЧАБ (количество абонентов) и КОДРАИОНА. В отношении АБОНЕНТЫ шесть доменов: РАСЧНОМ (расчетный номер), КОДРАИОНА, НОМЕРАБ (номер телефона абонента), КОЛАП (количество телефонных аппаратов), КАТЕГОР (категория абонента), МПЛАТА (месячная абонентская плата). В отношении ТАРИФЫ два домена: КАТЕГОР (категория) и МИНПЛАТА (минимальная месячная абонентская плата).

Требуется составить RAP-программу для подсчета суммарной месячной абонентской платы по всем абонентам категории Q района Майами, имеющим более одного телефонного аппарата и месячную плату, равную ее минимальному значению для указанной категории.

20.6. Предположим БД, описанная в предыдущем упражнении, характеризуется следующими числовыми параметрами: число записей (кортежей) в отношении АДРЕСА равно 1000, в отношении АБОНЕНТЫ равно 1 000 000, а в отношении ТАРИФЫ составляет 13. Кроме того, положим  $k=5$  и будем считать, что все абоненты Майами относятся к одному району. Требуется определить, за какое количество циклических просмотров БД может быть обработан сформулированный в предыдущем упражнении запрос.

20.7. Допустим, что требуется выполнить совместную обработку отношений, изображенных на рис. 20.4, для маркирования кортежей с информацией о всех служащих всех подразделений, в которых объем проданной продукции

(VOL) хотя бы одного из выпускаемых видов изделий меньше 1000. Как воспользоваться командой GET-FIRST-MARK для уменьшения времени выполнения при использовании команды CROSS-SELECT?

20.8. В условиях предыдущего примера предположим дополнительно, что количество записей в отношении ITEM, удовлетворяющих сформулированному требованию, равно 200. Пусть они относятся к 10 различным подразделениям, а все отношение ITEM распределено по двум секциям процессора RAP. Насколько сократится время обработки при использовании операции проектирования по сравнению с использованием команды совместной обработки CROSS-SELECT?

## ЛИТЕРАТУРА

1. Ozkarahan E. A., Schuster S. A., Smith K. C., An Associative Processor for Data Base Management, Proceedings of the 1975 NCC, Montvale, N. J., AFIPS, 1975, pp. 379—387.
2. Ozkarahan E. A., Schuster S. A., Smith K. C., A Data Base Processor, CSRG-43, University of Toronto, Toronto, 1974.
3. Schuster S. A., Ozkarahan E. A., Smith K. C., A Virtual Memory System for a Relational Associative Processor, Proceedings of the 1976 NCC, Montvale, N. J., AFIPS, 1976, pp. 855—862.
4. Schuster S. A., Ozkarahan E. A., Smith K. C., The Case for a Parallel-Associative Approach to Data Base Machine Architectures, Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks, Springfield, VA, National Technical Information Service, 1976, pp. 365—375.
5. Ozkarahan E. A., Schuster S. A., Sevcik K. C., Performance Evaluation of a Relational Associative Processor, *ACM Transactions on Database Systems*, 2(2), 175—195 (1977).
6. Ozkarahan E. A., Sevcik K. C., Analysis of Architectural Features for Enhancing the Performance of a Database Machine, *ACM Transactions on Database Systems*, 2(4), 297—316 (1977).
7. Sadowski P. J., Schuster S. A., Exploiting Parallelism in a Relational Associative Processor, Proceedings of the Fourth Workshop on Computer Architecture for Non-Numeric Processing, New York, ACM, 1978, pp. 99—109.
8. Schuster S. A., Nguyen H. B., Ozkarahan E. A., Smith K. C., RAP.2 — An Associative Processor for Databases and Its Applications, *IEEE Transactions on Computers*, C-28(6), 446—458 (1979).
9. Orlazer K., Ozkarahan E. A., Smith K. C., RAP. 3 — A Multi-Microprocessor Cell Architecture for the RAP Database Machine, Proceedings of the International Workshop on High-Level Language Computer Architecture, University of Maryland, College Park, MD, 1980, pp. 108—129.
10. Codd E. F., A Relational Model of Data for Large Shared Data Banks, *Communications of the ACM*, 13(6), 377—387 (1970).
11. Date C. J., An Introduction to Database Systems, Reading, MA, Addison-Wesley, 1975.

---

## ГЛАВА 2:

### ДРУГИЕ МАШИНЫ БАЗЫ ДАННЫХ

Реляционный ассоциативный процессор (RAP) является собой один из возможных примеров применения принципа ассоциативности для обработки информации, хранимой в БД. В этой главе рассматриваются две вычислительные машины, реализующие этот принцип и предназначенные для использования в качестве специализированных машин базы данных.

#### ПРОЦЕССОР CASSM

Процессор CASSM (context addressed segment sequential memory), или процессор с контекстуально адресуемой сегментированной последовательной памятью, имеет много общего с процессором RAP. Программное моделирование процессора CASSM было осуществлено в Университете шт. Флорида. К настоящему времени выполнена и аппаратная реализация одной ячейки этой системы. Ниже перечисляются основные характеристики подобия и различия процессоров CASSM и RAP.

1. Принципы организации систем CASSM и RAP почти полностью идентичны. Процессор CASSM состоит из контроллера и набора ячеек, каждая из которых имеет запоминающее устройство в виде вращающегося носителя.

2. Хранимые в процессоре CASSM данные организованы в иерархические (древовидные) структуры.

3. Все данные хранятся вместе со своими описаниями (т. е. имеют теги). Записи одного и того же файла не обязательно имеют одну и ту же длину или формат.

4. Команды в процессоре CASSM поступают непосредственно из БД, а не из обслуживающей машины (ведущей ЭВМ).

#### ОРГАНИЗАЦИЯ ДАННЫХ В ПРОЦЕССОРЕ CASSM

Логическая структура БД в процессоре CASSM может быть представлена одним или несколькими иерархическими деревьями связи данных. В качестве примера рассмотрим БД грузовых

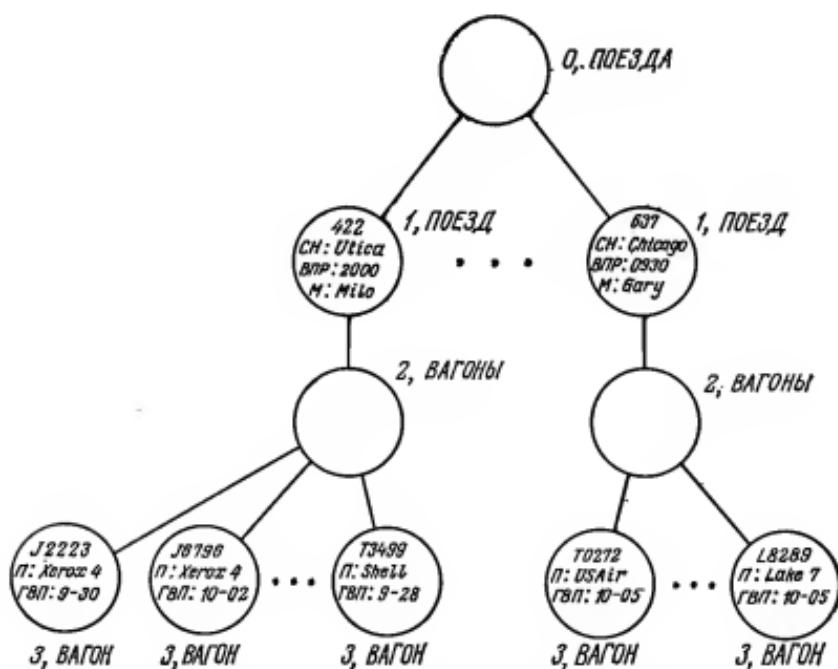


Рис. 21.1. Логическая структура файла в процессоре CASSM.

СН — станция назначения, М — местонахождение, П — получатель, ГВП — гарантированное время прибытия по расписанию, ВПР — время прибытия по расписанию.

железнодорожных перевозок. Пусть, в частности, один из файлов БД содержит информацию о составах и грузах, находящихся в пути. Каждый конкретный поезд в БД представляется отдельной записью, которая содержит следующие данные: номер железнодорожного состава, станцию назначения, дату и время прибытия по расписанию, текущее местонахождение и информацию маршрутных листов отдельных вагонов данного состава. Последняя может состоять из инвентарного номера вагона, идентификатора получателя и гарантированной даты прибытия. Логическая структура такого файла в CASSM может быть представлена в форме, показанной на рис. 21.1<sup>1)</sup>.

На вращающихся носителях памяти процессора CASSM данные представлены отдельными словами, состоящими из 40 двоичных разрядов. Форма доступа — последовательная по разрядам. Два разряда из 40 используются для обеспечения «внут-

<sup>1)</sup> Несущественное отличие структуры, изображенной на рис. 21.1, и соответствующей ей структуры записей в памяти от приведенного здесь описания заключается в отсутствии информации о дате прибытия по расписанию.—  
Прим. перев.

Таблица 21.1. Типы слов, используемых в процессоре CASSM

Тип слова	Шифр типа слова	Тег	Состояние	Данные (32 двоичных разряда)
Ограничитель	D	000	M, H, С	ИМЯ-УРОВЕНЬ, BSTK, S, Q
Имя-значение	N	001	M, H, С	ИМЯ, ЗНАЧЕНИЕ
Указатель	P	010	M, H, С	ИМЯ, УКАЗАТЕЛЬ
Строка	S	011	M, H, С	4 символа
Команда	I	100	TP, А	Команда или литерал
Операнд	O	101	LN, F	32-битовое значение
Защита доступа	L	110		
Данные, подлежащие исключению	E	111	WD	

*Примечания.*

М — бит совпадения,  
Н — бит сохранения,

С — бит маркирования выводимых данных,

С — бит выделения данных, которые могут отмечаться в результате выполнения операции поиска,

Q — бит промежуточного маркирования данных,  
A — бит активации,

F — флаг конца списка,

ИМЯ — имя единицы информации предметной области (16 разрядов),

УРОВЕНЬ — порядковый номер уровня в иерархии (8 разрядов),

BSTK — стек результатов операции поиска (6 разрядов),

ЗНАЧЕНИЕ — значение в двоичной форме (16 разрядов),

УКАЗАТЕЛЬ — номер записи,

TP — указатель команды (слово является командой или литералом),

LN — порядковый номер операанда в стеке или очереди (2 разряда),

WD — указатель конца файла, временного характера данных или «мусора», не подлежащего хранению.

ренных потребностей» ячейки, три разряда — для записи тега данных, три других разряда являются индикаторами состояния, а в оставшихся 32 разрядах обычно содержатся данные. Допускается использование слов восьми типов (см. табл. 21.1).

Для определения каждого узла каждого дерева БД используется слово *ограничитель*. Оно определяет имя узла в БД (например, «железнодорожный состав») и его уровень в иерархии. Основные операции системы CASSM выполняются при использовании следующих разрядов слова: BSTK, S и Q. Группа разрядов BSTK используется подобно маркерным разрядам кортежей в системе RAP, хотя и представляет собой память стекового типа. Разряд Q является индикатором, определяющим данные, подлежащие обработке в предстоящей поисковой операции. Перед началом этой операции CASSM-программа устанавливает значение этого индикатора. В частности, производится предварительная установка в 1 разряда Q для намеченных к обработке узлов БД.

Функциям разряда S в ограничителе нет аналога в системе RAP. Реляционный принцип организации данных в RAP одно-

значно определяет, что по завершении поиска подлежит маркировке (а именно, кортежи, удовлетворяющие заданному условию). В случае же иерархической организации данных, принятой в CASSM, ответ на подобный вопрос не столь очевиден. Пусть, например, выполняется поиск вагона с инвентарным номером J6796. При этом остается неясным, следует ли отметить запись о самом вагоне, о поезде, в состав которого он входит, или всю группу поездов. Двоичная единица в разряде S, устанавливаемая до начала поиска, используется для указания, какой именно узел (узлы) подлежит маркировке после успешного завершения поиска. Например, если имеется необходимость отметить поезд, в состав которого входит вагон J6796, то следует установить в 1 значение разряда S во всех узлах типа 1, ПОЕЗД.

Слово *имя-значение* используется для определения каждого поля в каждом узле структуры БД. В этом слове содержится имя поля (в системе CASSM имя поля включается в слова «ограничитель» и «имя-значение», подобно тому как имена полей включаются в записи системы RAP) и его значение (в виде дополнительного двоичного 16-разрядного кода). Благодаря этому информация о каждом поле в БД является самоопределяемой. Поэтому система допускает произвольную последовательность записей данных об отдельных полях или узлах БД, отсутствие отдельных полей в однотипных записях и многократное повторение поля в одной и той же записи.

Слово *указатель* применяется для различных целей. Его можно использовать в тех случаях, где допустимо применение слова «имя-значение» при условии, что вместо значения используется адрес или указатель. Слово «указатель» служит для представления полей с нечисловым содержимым, а именно полей, в которых находится одно или несколько слов типа «строка»; при этом слово «указатель» определяет отдельное поле и задает адрес первого слова «строка». Словом «указатель» можно пользоваться также в целях уменьшения избыточности (в частности, если разные поля имеют одно и то же нечисловое содержимое, то можно ограничиться одним таким полем, обращаясь к нему по мере необходимости посредством слова «указатель»). С помощью слова «указатель» иерархическое представление данных переводится в сетевое представление или ориентированный граф.

Иерархические структуры данных записываются в запоминающие устройства ячеек системы CASSM в соответствии с принципом так называемой *линейной развертки*: сверху вниз и слева направо. Иллюстрацией сказанному может служить представленное на рис. 21.2 содержимое слов «ограничитель» и «имя-значение». В этих словах указано содержимое только

Ограничитель	C	Имя файла/записи	Уровень в иерархии	Разуль-таты поиска	S	Q
Имя-значение	C	Имя поля	Значение поля			

Рис. 21.2. Слово «ограничитель» и слово «имя-значение».

тех полей, которые необходимы для дальнейших рассуждений. Разряд C (Collection) полей используется для маркировки той информации в БД, которая будет вводиться в ведущую машину.

В табл. 21.2 показана физическая структура записей, соответствующая иерархической БД, изображенной на рис. 21.1. Для простоты приведено только несколько слов структуры и игнорируется возможность хранения в словах «имя-значение» нечисловых данных.

#### НАБОР КОМАНД СИСТЕМЫ CASSM

Набор, комаид, относящихся к процессору CASSM, не является обшириым, однако многие комаиды позволяют реализовать большое разнообразие операций того или иного типа посредством задания в комаиде необязательных параметров. К четырем основным комаидам обработки данных относятся следующие: маркировка ограничителя (DMK), поиск данных по некоторому условию (QSR), определение местоположения данных (FND) и определение состояния данных (FNS).

Комаида DMK используется для просмотра записей БД с целью:

1) маркировки тех записей, которые предназначаются для следующего поиска (т. е. целью просмотра является установка в 1 разрядов Q в части записей БД), или

2) маркировки тех узлов (ограничителей), в которые должны быть помещены результаты последующего поиска (т. е. целью просмотра является установка разрядов S в соответствующих записях БД).

Так, комаида DMK W 1,TRAIN выделяет из всех записей БД (признак W) ограничители на первом уровне с именем TRAIN и устанавливает в 1 их разряды S и Q. Комаида DMK D Q\*,\* устанавливает в 1 только разряд Q во всех узлах всех уровней (\*) с любыми именами (\*), если эти узлы подчинены (признак D—descendent) хотя бы одному узлу с уже установленным в 1 разрядом Q.

Таблица 21.2. Пример линейного представления иерархической структуры БД в процессоре

Структура записей							
C	Тег	Имя	Значение	Уровень	BSTK	S	Q
0	D	поезда		0	000000	0	0
0	D	поезд		1	000000	0	0
0	N	инвентарный номер	422				
0	N	станция назначения	Utica				
0	N	ожидаемое время прибытия	2000				
0	N	местонахождение	Milo				
0	D	вагоны		2	000000	0	0
0	D	вагон		3	000000	0	0
0	N	инвентарный номер	J2223				
0	N	идентификатор получателя	Xerox4				
0	N	гарантиированная дата	0930				
0	D	вагон		3	000000	0	0
0	N	инвентарный номер	J6796				
0	N	идентификатор получателя	Xerox4				
0	N	гарантиированная дата	1002				
0	D	вагон		3	000000	0	0
0	N	инвентарный номер	T3499				
0	N	идентификатор получателя	Shell				
0	N	гарантиированная дата	0928				
0	D	поезд		1	000000	0	0
0	N	инвентарный номер	637				
0	N	станция назначения	Chicago				
0	N	ожидаемое время прибытия	1130				
0	N	местонахождение	Gary				
0	D	вагоны		2	000000	0	0
0	D	вагон		3	000000	0	0
0	N	инвентарный номер	T0272				
0	N	идентификатор получателя	USAir				
0	N	гарантиированная дата	1005				
0	D	вагон		3	000000	0	0
0	N	инвентарный номер	L8289				
0	N	идентификатор получателя	Lake7				
0	N	гарантиированная дата	1005				

Команда QSR подобна команде SELECT процессора RAP. Поиск проводится по всем узлам с маркером Q (т. е. разряд которых установлен в 1). Одной из функций команды QSR является выполнение логической операции ИЛИ над результатами обработки всех узлов поддерева с маркером Q, после чего определяется местоположение узлов с маркером S. (Последние находятся на том же или более высоком уровне иерархии.) Результат упомянутой операции ИЛИ может быть либо непосредственно переслан в стек BSTK соответствующего узла с маркером S, либо путем выполнения логической операции И или ИЛИ над этим результатом и битом на вершине стека BSTK. В процессе поиска могут выполняться требуемые сравнения (на равенство, на превышение и т. п.).

Команды FND и FNS используются для модификации данных и их маркировки с целью последующего вывода. Параметры команд определяют условия отбора и предпринимаемые действия. Условия отбора могут учитывать состояние данных (которое может быть выражено значением содержимого разряда Q), значение бита на вершине стека BSTK или значение данных. Действия могут заключаться в маркировке данных для последующего вывода (путем установки в 1 разряда С), модификации данных в указанных полях или выполнении таких операций над содержимым группы выделенных полей, как вычисление суммы всех значений, количества последних, их максимальной или минимальной величины.

Функционирование команд может быть проиллюстрировано примером запроса к БД. В частности, для БД, структура которой представлена на рис. 21.1 и в табл. 21.2, запрос «Найти поезда, в состав которых входят вагоны с гарантированной датой прибытия до 6 октября (1006) или гарантированной датой прибытия 10 октября (1010)» выполняется с использованием следующей последовательности команд:

```
DMK W 1,train
DMK D Q**.
QSR <,N;PUB;IM(gad:1006)
QSR =,N;ORB;IM(gad:1010)
FND B N,iden;ORS C;LDP, WDR,CP,WTC
```

Первые две команды этой программы были рассмотрены выше. Следующая команда QSR просматривает все вершины дерева с установленным в 1 разрядом Q (т. е. все поезда и вагоны) в поисках тех из них, в которых значение слова «имя-значение» с именем gad (гарантированная дата прибытия) меньше 1006. Результаты поиска по вершинам, подчиненным вершинам с маркерами S, объединяются логической операцией ИЛИ и затем посыпаются (операнд PUB) в стек BSTK вершин

с маркером S. Вторая команда QSR выделяет слова «имя-значение» с именем gad и значением 1010. Результаты поиска по вершинам, подчиненным вершине с маркером S, объединяются логической операцией ИЛИ между собой и битом на вершине стека BSTK и замещают его (операнд ORB).

По команде FND осуществляется поиск слов «имя-значение» с именем iden. Если биту на вершине стека BSTK предшествующего ограничителя присвоено значение 1, то содержимое разряда С замещается на результат логической операции ИЛИ над ним и 1 (операнд ORS). Согласно операнду LDP (load controller processor status word), осуществляется загрузка PSW, в результате чего содержимое вершин с маркерами С пересыпается в ведущую машину или управляющий процессор (операнд CP) в произвольном (с точки зрения следования слов) порядке (операнд WDR). При этом запрещается изменение PSW до окончания передачи содержимого всех С-маркированных вершин (операнд WTC). Функционирование процессора CASSM регулируется последовательностью команд и значением PSW.

В процессоре CASSM имеются средства межфайловой маркировки. Эти возможности реализуются с помощью operandов XWR и XRW команд FND и FNS. Каждая ячейка системы CASSM располагает 1-разрядной памятью произвольного доступа. При выполнении указанных операций для всех  $p$  ячеек — «участниц» операции создается общая 1-разрядная память произвольного доступа объемом  $p$ . По команде FND с операндом XWR в эту память записываются значения всех данных, удовлетворяющих критериям отбора. При этом, если значение данных, представленное числом 5, удовлетворяет критериям отбора, пятому разряду указанной памяти присваивается единичное значение. (Подобный режим работы предполагает предварительное кодирование значений в данных элементами множества целых чисел 0, ...,  $p$ .) После указанной операции в упомянутой памяти из 1-разрядных элементов оказываются отмеченными все значения данных, удовлетворяющие установленным критериям отбора. Теперь может быть применена команда FND с операндом XRW для маркировки значений данных в другом файле, если единичное значение присвоено тому биту 1-разрядной памяти, который соответствует заданному значению.

#### ВЫПОЛНЕНИЕ CASSM-ПРОГРАММ

Необычной характеристикой системы CASSM, составляющей в то же время главное отличие процессора CASSM от процессора RAP, является хранение команд CASSM в базе данных. В соответствии с табл. 21.1 команды можно рассматривать

как особый вид хранимой в БД информации, т. е. как одну из разновидностей структур данных. Программа для CASSM может быть представлена в виде дерева команд в БД, где поддеревья (вводимые словами «ограничитель») аналогичны подпрограммам и блокам, вводимым служебным словом begin. В отличие от процессора RAP здесь команды не поступают из ведущей машины. В результате процессор CASSM может использоватьсь как автономная система, где в основе инициирования команд лежат модифицирующие операции над данными.

В соответствии с табл. 21.1 в коде каждой команды имеется так называемый *разряд активности* (разряд А). Ячейки CASSM сканируют свою память, и каждая встречающаяся при этом команда выполняется, если ее разряд А установлен в 1. После выполнения команды разряд А устанавливается в 0, т. е. сбрасывается. Помимо рассмотренных выше команд в системе CASSM предусмотрена группа команд управления выполнением программы. Эти команды могут активировать иdezактивировать другие команды. Так, команда ACT(ACTivate) устанавливает в 1 разряд А в указанной группе команд. Команда DAC (DeACTivate) выполняет обратное действие. Команда JMP сбрасывает разряд А во всех командах БД, а затем устанавливает его в 1 в предписанном множестве команд. Команда ACS подобна команде ACT, но осуществляет также загрузку содержимого разряда S в ограничителях, предшествующих командам в дереве программы, в стек BSTK этих ограничителей. Команда POP восстанавливает содержимое разряда S, извлекая его из стека.

## МАШИНА БАЗЫ ДАННЫХ

Совершенно другой подход к реализации принципов ассоциативности применен при проектировании машины базы данных DBC (database computer) в Университете шт. Огайо. Основные отличия машины DBC от процессоров RAP и CASSM заключаются в следующем:

1. По характеру элементов DBC является более разнородной системой по сравнению с системами RAP или CASSM. Хотя машина DBC содержит ячейки ассоциативного поиска, подобные имеющимся в процессорах RAP и CASSM, они составляют незначительную часть элементов логической обработки данных.

2. В системе DBC каждая ячейка ассоциативного поиска оперирует не в своем единственном запоминающем устройстве, а в пределах пула (объединения) подобных устройств, которые имеют значительный объем памяти. В результате для системы DBC характерна пониженная величина отношения числа

компонентов для ассоциативной обработки к числу адресуемых элементов памяти.

3. В системе DBC, как и в традиционных СУБД, реализуемых программным способом, для доступа к данным используется система индексов. Однако для обработки как данных, так и самих индексов применяются ассоциативные методы.

4. В архитектуре DBC не заложено какой-либо определенной модели БД (реляционной, иерархической или сетевой). Поэтому модели обрабатываемых данных здесь примитивнее, чем в системах RAP или CASSM, а возможности команд более ограничены.

5. Неотъемлемым компонентом DBC являются средства защиты от несанкционированного доступа к данным.

#### СТРУКТУРА СИСТЕМЫ DBC

Согласно рис. 21.3, структура системы DBC представляется в виде двух связанных замкнутых цепочек блоков обработки. Эти цепочки совместно используют процессор общего управления и анализа команд данной СУБД (DBCCP — database command and control processor). Обычая последовательность выполнения операций состоит в том, что сначала эти операции выполняются элементами цепочки обработки информации о структуре размещения данных, а затем — элементами цепочки выполнения операций над данными в основной памяти. В каждой из этих двух замкнутых цепочек могут быть выделены отдельные субблоки обработки (процессоры или наборы процессоров). В результате обработка групповых операций в этой системе выполняется в форме, типичной для копиейерных систем: в один и тот же момент времени эти операции могут обслуживаться разными субблоками обработки.

Побуждающим мотивом разработки системы DBC явилось интуитивное предположение о том, что системы с регулярной структурой, такие, как RAP или CASSM, не будут рентабельны в условиях больших БД ближайшего будущего. В поддержку такого утверждения можно привести два аргумента. Во-первых, несмотря на использование методов частично ассоциативного доступа в RAP и CASSM, отношение числа обрабатывающих элементов к числу элементов памяти в этих системах остается достаточно высоким. При работе с БД большого объема (порядка  $10^{10}$  байт и выше) при выполнении реляционным ассоциативным процессором операции над отдельным отношением может оказаться задействованной лишь небольшая часть ячеек RAP (при общем количестве отношений, исчисляемом сотиями). В результате обрабатывающие средства системы оказываются загруженными далеко не полностью.

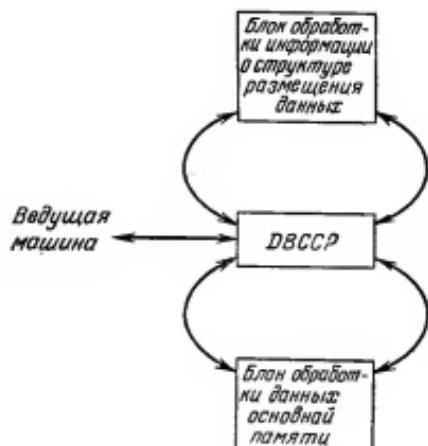


Рис. 21.3. Две цепочки блоков обработки машины DBC.

Второй аргумент в пользу применения системы DBC вытекает из стоимостью оценки памяти. Память систем RAP и CASSM основана на вращающихся носителях типа магнитных дисков с фиксированными головками или их электронных эквивалентах (память на магнитных цилиндрических доменах или элементах с зарядовой связью). Стоимость памяти такого типа на порядок или более превышает стоимость памяти, в которой реализован традиционный способ хранения информации на дисках с перемещаемыми головками. В связи с этим

структуря процессора DBC должна была представлять собой машину базы данных с ассоциативным доступом на основе дешевых накопителей на дисках с перемещаемыми головками, лишеннюю в то же время их очевидных недостатков.

В системе DBC вначале в работу вступают блоки цепочки обработки информации о структуре размещения данных, а затем выполняются операции над данными основной памяти. Однако мы начнем рассмотрение с операций над данными основной памяти, поскольку именно эти операции определяют само существование цепочки обработки информации о структуре размещения данных.

#### ОПЕРАЦИИ НАД ДАННЫМИ В ОСНОВНОЙ ПАМЯТИ В СИСТЕМЕ DBC

Основные элементы цепочки блоков, выполняющих операции над данными в основной памяти, показаны на рис. 21.4, а. В состав этой подсистемы входят устройство управления и  $n$  «сборщиков информации на дорожках» (TIP-элементы<sup>1</sup>), где  $n$  — число дорожек в одном цилиндре дисковой памяти, т. е. число головок считывания/записи. Каждый из элементов TIP представляет собой ассоциативный процессор, напоминающий ячейку системы RAP или CASSM. Он не имеет постоянной связи с каким-либо единственным вращающимся носителем или до-

<sup>1</sup> TIP — аббревиатура, образованная от английских слов track information processors. — Прим. перев.

режкой. Данные в системе фактически могут храниться на многих отдельных томах дисковой памяти со стандартными механизмами перемещения головок считывания/записи. Посредством устройства управления механизмом доступа к памяти и набора селекторов механизмов доступа каждый процессор обработки

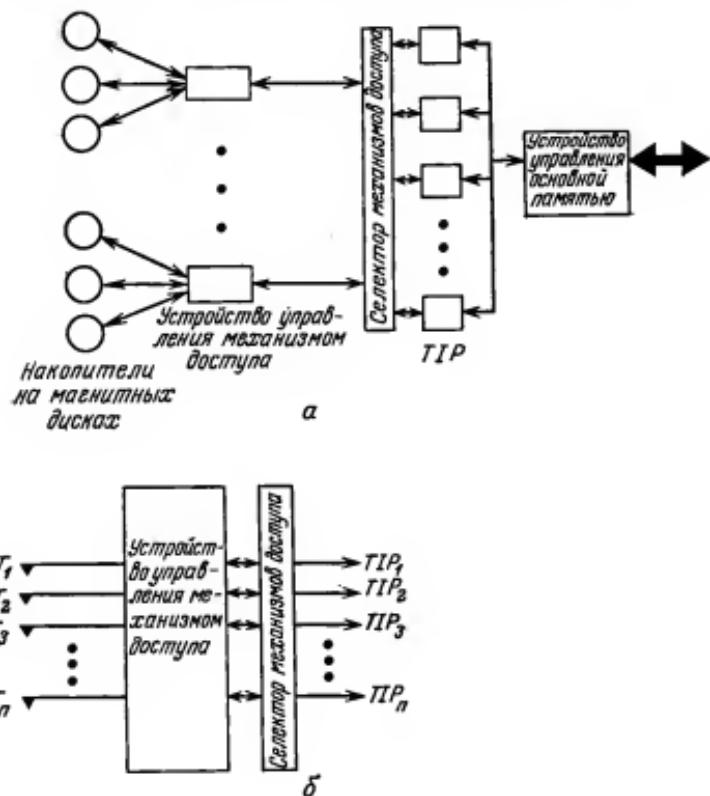


Рис. 21.4. Элементы основной памяти машины DBC.  
а — функциональная взаимосвязь элементов; б — модифицированное устройство управления на магнитных дисках.

TIP может быть соединен с соответствующей дорожкой любого типа. Например, если в системе имеется 100 томов дисковой памяти, то процессор TIP<sub>1</sub> может быть подсоединен к первой дорожке любого из этих 100 томов.

Основной недостаток такого технического решения заключается в том, что при использовании стандартных устройств управления накопителями на магнитных дисках с перемещаемыми головками считывание данных осуществляется поразрядно,

поочередно с отдельных дорожек. Такой режим передачи данных обесценивает принцип закрепления за каждой дорожкой своего процессора (TIP), поскольку информация в каждый данный момент времени могла бы поступать только к одному такому процессору. Для эффективной работы системы необходимо, чтобы имелась возможность выбора дисковода, установки головок считывания/записи на определенный цилиндр и одновременно записи или чтения информации всех дорожек этого цилиндра (т. е. одновременного использования всех головок считывания/записи). Оказывается подобное изменение режима работы возможно без коренной переделки устройства управления и акопителем на магнитных дисках. Этот новый способ и применен в системе DBC. Схематически связь процессоров обработки с головками считывания/записи изображена на рис. 21.4, б.

Рассмотрим последовательность действий системы при выполнении операций над данными основной памяти. Сначала производится выбор накопителя. Затем головки считывания/записи устанавливаются на требуемый цилиндр. После этого данные со всех дорожек этого цилиндра параллельночитываются в процессоры обработки TIP. По мере поступления данных через селектор механизмов доступа в процессоры обработки, последние выполняют анализ этих данных. Каждый из процессоров может передавать устройству управления основной памятью информацию о результатах выполнения операции поиска. Вследствие высокой плотности записи данных на диски, а также из-за технических затруднений, возникающих при попытке согласованного размещения нескольких головок на одной дорожке, в данной системе используются одни и те же головки как для записи, так и для считывания информации (в системе RAP.1 для записи и считывания используются разные головки). Поэтому для модификации записей требуется не менее двух оборотов диска.

При указаний организации процессора основной памяти ассоциативность обработки данных достигается в пределах отдельных цилиндров. Серьезным препятствием для эффективного использования системы является чрезмерно большое время, необходимое для поиска по всей БД, если отсутствует уточняющая информация о физическом расположении данных. С целью преодоления этой проблемы предусмотрено следующее: при поступлении в процессор DBCCP запроса на выполнение операции устройства управления основной памятью получает также список номеров цилиндров (томов). Эта информация сужает область поиска, указывая только те части БД, где могут находиться представляющие интерес данные. Помимо этого прибегают к рациональной заблаговременной группировке данных

в БД (в частности, таким образом, чтобы данные, которые могут служить предметом одновременных поисков, располагались на одном цилиндре или малом числе цилиндров). Благодаря этому при выполнении стандартных операций в DBC удастся ограничиться просмотром всего лишь одного или нескольких цилиндров, что существенно для преодоления основных трудностей, связанных с использованием накопителей на магнитных дисках с перемещаемыми головками. Кроме этого, устройство управления основной памятью может поддерживать режим конвейерной обработки команд. А именно, в то время, когда процессоры обработки ТИР выполняют анализ данных, полученных с некоторого цилиндра, может быть выполнена перестановка головок считывания/записи или выбран другой накопитель на магнитных дисках в порядке подготовки к выполнению следующей операции.

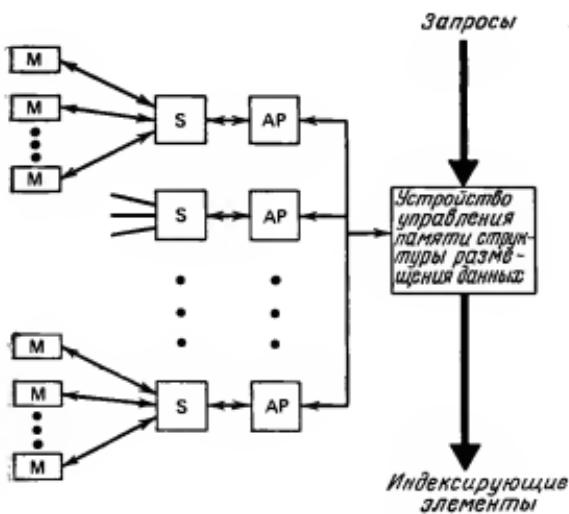
Эффективность операций над данными основной памяти в DBC существенно зависит от возможности предварительного определения подмножества цилиндров, обработкой которых можно ограничиться. Именно для решения этой задачи в систему DBC введена цепочка блоков обработки информации о структуре размещения данных.

#### ОБРАБОТКА ИНФОРМАЦИИ О СТРУКТУРЕ РАЗМЕЩЕНИЯ ДАННЫХ В DBC

Цепочка блоков обработки информации о структуре размещения данных в памяти выполняет роль механизма индексирования. Но в отличие от стандартных схем индексирования, которыми обеспечивается доступ к конкретной записи на конкретной дорожке заданного цилиндра, данная схема индексирования позволяет определять только соответствующие цилиндры. Окончательный же ассоциативный поиск данных выполняется посредством операций над данными в основной памяти.

Основными элементами цепочки блоков обработки информации о структуре размещения данных в основной памяти БД являются память структуры размещения данных и ее устройство управления (рис. 21.5). Организация этой памяти подобна организации основной памяти хранения данных, отличия этих двух запоминающих устройств носят технологический характер. Основное функциональное назначение памяти структуры размещения данных — определение для любого запроса к БД множества цилиндров, на которых расположено хотя бы по одной записи, удовлетворяющей параметрам запроса. Сам запрос представляется в виде дизъюнкции ключевых предикатов, под которыми понимают логические выражения, состоящие из имени атрибута, оператора отношения и значения (например, заработок  $> 20\ 000$ ).

Доступ к памяти структуры размещения данных организован также на основе ассоциативных способов обработки: в состав обслуживающего ее процессора входит набор блоков ассоциативных процессоров (AP). Подобно процессору основной памяти, и здесь связь между ассоциативными процессорами и соответствующими запоминающими устройствами на вращающихся носителях не является фиксированной, раз и навсегда заданной. С помощью переключателей каждый ассоциативный процессор



лелнзм обработки информации в памятн структуры размещения данных. Для обеспечения параллелнзма информацию в памятн структуры размещения данных следует располагать таким образом, чтобы индексы цилнндов для определенных атрибутов по возможностн распределялись по модулям памяти, связанным с различными процессорами.

#### ПРЕДСТАВЛЕНИЕ ДАННЫХ В МАШИНЕ DBC

Записи, храннмые в основной памятн БД, состоят из трех частей: идентифицирующей части, набора пар «атрибуты-значение» и информационной части. Формат записи представлен на рис. 21.6. Записи, которые вероятнее всего будут обрабатывать-



Рис. 21.6. Формат записи в машине DBC.

C — номер кластера; S — идентификатор блока санкционированного доступа; K — число ключевых слов в записи; A<sub>i</sub> — номер атрибута; L<sub>i</sub> — длина поля V<sub>i</sub>; V<sub>i</sub> — значение ключевого слова; A<sub>1</sub><...<A<sub>k</sub>.

ся совместно, размещают на возможно меньшем количестве цилнндов. Это обеспечивается специальным механизмом группировки. При вводе записи для хранения в машину DBC из ведущей машины должен также поступать идентификатор конкретной группы — кластера — данных, к которой должна быть «принесана» эта запись. Кластер можно представить себе в виде некоторого файла, хотя и не обязательно. Каждая запись включает идентификатор кластера, которому она принадлежит.

В идентифицирующей части записи содержится также указатель блока санкционированного доступа, которому принадлежит эта запись. Пусть, например в системе учета кадров необходимо принять меры по защите некоторых записей кластера от доступа каких-либо пользователей или программ. Кластер может содержать данные как о штатных сотрудниках, так и о совместителях, т. е. сотрудниках с неполной рабочей неделей (работающих на полставки). Предположим, задача заключается в том, чтобы разрешить доступ отдельным пользователям БД лишь к информации о совместителях. Для этого достаточно ввести два блока санкционированного доступа и организовать работу таким образом, чтобы группа пользователей имела доступ лишь к одному из этих блоков.

Вторая часть записи состоит из пар «атрибуты-значение». Каждую такую пару называют *ключевым словом*, являющимся

по сути самоопределяемыми данными. Идентификатор ключевого слова является номером атрибута. Перечисление атрибутов в записи упорядочено по убыванию номеров, хотя в самой последовательности номеров допустимы пробелы. Такое упорядочение ключевых слов в записи позволяет упростить процедуру поиска, ибо информация о запросе, передаваемая в процессор основной памяти (триплеты «атрибут-отношение-значение»), также должна быть упорядочена по убыванию номеров атрибутов.

Возможности ассоциативной обработки в БД распространяются не на всю структуру записи, а лишь на данные, зафиксированные в парах «атрибут-значение». Данные, не подлежащие ассоциативной обработке, размещаются в информационной части записи. Они воспринимаются системой как последовательности символов, которые могут быть записаны в БД или извлечены из нее.

#### НАБОР КОМАНД, ИСПОЛЬЗУЕМЫЙ В МАШИНЕ DBC

По сравнению с процессорами RAP и CASSM в машине DBC используется простой набор команд. Команда поиска и извлечения данных имеет следующий формат:

RETRIEVE <идентификатор файла, список полей, условие отбора записей>

В списке полей перечисляются имена всех атрибутов, значения которых должны быть получены в результате поиска. Условие отбора значений представляется дизъюнкцией ключевых предикатов, состоящих из имени атрибута, оператора отношения и значения.

К числу основных команд, используемых в системе DBC, относятся также следующие команды: DELETE — исключение всех записей, которые удовлетворяют условию отбора; INSERT — добавление записи в БД (задается и обрабатывается информация о кластерах); REPLACE — задание списка ключевых слов (пар «атрибуты-значение») с целью замены ими текущих значений атрибутов во всех записях, удовлетворяющих условию отбора.

#### БЛОКИ ОБРАБОТКИ МАШИНЫ DBC

Основными блоками машины DBC, в которых выполняется ассоциативная обработка, являются основная память и память структуры размещения данных. Помимо этого, в системе DBC имеется несколько других блоков, участвующих в обслужива-

нни запросов к БД. Полная структура DBC, включающая эти блоки, показана на рис. 21.7.

Наиболее удобный способ ознакомления с взаимодействием между отдельными блоками заключается в рассмотрении последовательной обработки запроса блоками двух упомянутых

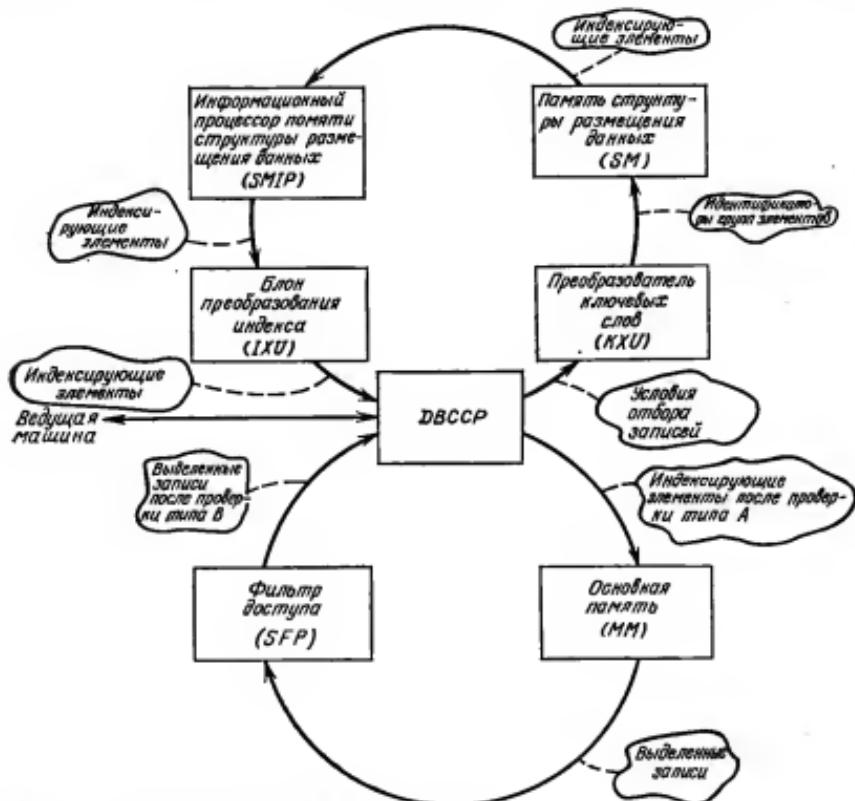


Рис. 21.7. Обработка запроса в машине DBC.

ранее замкнутых цепочек, образующих DBC. При этом следует учитывать, что обработка запросов в обоих цепочках имеет в значительной степени конвейерный характер, а именно: в то время как параметры некоторого запроса обрабатываются каким-либо одним блоком (элементом цепочки), в остальных блоках этой цепочки может выполняться обработка других запросов.

При поступлении запроса в DBCCP (*процессор общего управления и анализа команд системы DBC*) условие отбора записей передается в преобразователь ключевых слов (*keyword transformation unit — KXU*). Выше отмечалось, что память

структуры размещения данных служит для определения местоположения небольшого подмножества записей основной памяти, в котором должен быть проведен поиск. Блок КХУ выполняет аналогичные функции, но уже по отношению к самой памяти структуры размещения данных. В указанном блоке вырабатывается информация для процессора памяти структуры размещения данных о группах элементов в этой памяти, которые подлежат ассоциативной обработке для нахождения данных о цилиндрах. В блоке КХУ имеется своя память со всеми атрибутами (именами полей или идентификаторами), определяющими БД системы DBC. Условия отбора записей формулируются в виде одного или нескольких атрибутов, задаваемых идентификаторами переменной длины. Преобразователь КХУ переводит каждое из этих имен в специальный идентификатор атрибута с постоянной длиной. Он также выделяет числовые значения из условий отбора записей (например, значение 20 000 из предиката ЗАРАБОТОК > 20 000, построенного с использованием соответствующего ключевого слова) и передает пары управляющих данных — идентификатор атрибута в виде числа и выделенное значение — процессору памяти структуры размещения данных. Каждая из этих пар называется идентификатором группы, поскольку она определяет, какие именно модули составной памяти, связанные с каждым из ассоциативных процессоров памяти структуры размещения данных, содержат соответствующую информацию об индексах. Так функционирует в памяти структуры размещения данных упоминавшийся выше механизм группировки.

Исходными данными для работы процессора памяти структуры размещения данных (рис. 21.5) является список групп элементов, соответствующий условиям отбора записей. Путем ассоциативного просмотра информации об индексах процессор преобразует этот список в список индексирующих элементов. Последние состоят из относительного номера цилиндра и номера блока защиты от несанкционированного доступа. Например, условие отбора записей

**ЗАРАБОТОК > 20 000 или ВОЗРАСТ = 65**

может привести к формированию списка из пяти индексирующих элементов.

Список индексирующих элементов передается в информационный процессор памяти структуры размещения данных (structure memory information processor — SMIP). Этот процессор выполняет поэлементную проверку списка на пересечение и сокращает размер списка путем исключения из него элементов с дублирующими номерами цилиндров.

Процессор SMIP пересыпает получившийся список в блок преобразования индекса (index translation unit — IXU). В этом блоке выполняется простое преобразование индексирующих элементов. Дело в том, что блоки памяти структуры размещения данных оперируют не абсолютными, а относительными номерами цилиндров. Так, если файл расположен на пяти цилиндрах, при поиске записей этого файла будут использоваться относительные номера цилиндров 0—4. Применение относительных номеров обеспечивает более компактное хранение индексов в памяти структуры размещения данных. В блоке IXU имеется память соответствий между относительными номерами цилиндров в файлах и абсолютными номерами цилиндров. Блок IXU просто выполняет преобразование номеров цилиндров в списке индексирующих элементов из относительных в абсолютные.

Сформированный список индексирующих элементов передается обратно в процессор DBCCP. Этот процессор выполняет так называемую проверку типа А на наличие санкционированного доступа. Она основывается на анализе данных хранимых в DBCCP списков указателей блоков санкционированного доступа, с которыми могут работать отдельные пользователи. Получив из блока IXU список индексирующих элементов, процессор DBCCP выполняет сравнение номеров цилиндров в этом списке с номерами в списке цилиндров, доступных пользователю. Если при сопоставлении будут обнаружены индексирующие элементы, доступ к которым пользователю не разрешен, они исключаются из списка. (Проверку нельзя назвать полной, поскольку возможно несоответствие между размерами блоков санкционированного доступа и объемом памяти выделенных цилиндров. Дополнительная проверка на наличие санкционированного доступа выполняется на уровне процессора основной памяти во время считывания и обработки записей.)

Обработанный и, возможно, сокращенный список индексирующих элементов пересыпается процессору основной памяти. Последний, как отмечалось, обеспечивает ассоциативный доступ к данным на основе дисковой памяти с подвижными головками (рис. 21.4). Из блоков санкционированного доступа, к которым может обращаться данный пользователь, процессор основной памяти извлекает записи, удовлетворяющие условию отбора. Выделенные записи пересыпаются в фильтр доступа (security filter — SFP).

Фильтр доступа, представляющий собой еще одно средство контроля, выполняет проверку на наличие санкционированного доступа на уровне полей записи, именуемую проверкой типа В. В результате можно ограничить доступ пользователя не только к отдельным группам записей (путем указания блоков санкционируемого доступа), но и к отдельным типам полей в записях,

а также к данным, значения которых выходят за установленные границы. Если, например, в блок SFP передаются указания о том, что пользователь X не должен иметь доступа к атрибуту ЗАРАБОТОК, фильтр после получения записей из основной памяти, удаляет из них соответствующую информацию (пары «атрибут-значение»).

После всех указанных преобразований отфильтрованные данные — результат обработки запроса — передаются в основной процессор DBCCP.

#### ПРОИЗВОДИТЕЛЬНОСТЬ МАШИНЫ DBC

Как можно заметить, принципы построения системы DBC существенно отличаются от принципов, положенных в основу систем RAP и CASSM. Достижение высокой производительности системы DBC связывают со способностью этой машины базы данных выполнять в различных элементах двух ее цепочек одновременно обработку нескольких запросов. Сокращение расходов при использовании системы DBC предполагается достичь за счет относительно меньшего объема обрабатывающих элементов на единицу объема основной памяти.

Приближенная итоговая оценка производительности машины DBC может быть подсчитана следующим образом [5]. Процессор основной памяти при реализации ее на дисках с перемещаемыми головками выполняет поиск в пределах цилиндра за один оборот носителя. Допуская, что цилиндр содержит 40 дорожек, и учитывая, что в стандартных системах выполняется последовательное поразрядное считывание в каждый момент времени лишь с одной дорожки, получаем сорокакратное увеличение эффективности. Кроме того, поскольку, в то время как обрабатывается некоторый запрос в основной памяти, в процессоре памяти структуры может обрабатываться другой запрос, имеем еще двукратное увеличение эффективности. Наконец, вследствие конвейерного характера обработки (в обеих цепочках блоков обработки системы DBC могут одновременно обслуживаться четыре-пять запросов) можно рассчитывать на дополнительное повышение эффективности примерно в два раза. Таким образом, производительность системы DBC может превысить производительность традиционных СУБД в 160 и более раз.

#### ЛИТЕРАТУРА

1. Su S. Y. W. et al., The Architectural Features and Implementation Techniques of the Multicell CASSM, *IEEE Transactions on Computers*, C-28(6), 430—445 (1979).
2. Su S. Y. W., Emam A., CASDAL, CASSM's Data Language, *ACM Transactions on Database Systems*, 3(1), 57—91 (1978).

3. Su S. Y. W., Cellular Logic Devices, Concepts and Applications, *Computer*, 12(3), 11—25 (1979).
4. Lipovski G. J., Architecture Features of CASSM, A Context Addressed Segment Sequential Memory, Proceedings of the Fifth Annual Symposium on Computer Architecture, New York, ACM, 1978, pp. 31—43.
5. Banerjee J., Hsiao D. K., Kannan K., DBC—A Database Computer for Very Large Databases, *IEEE Transactions on Computers*, C-28(6), 414—429 (1979).
6. Kerr D. S., Database Machines with Large Content-Addressable Blocks and Structural Information Processors, *Computer*, 12(3), 64—79 (1979).
7. Banerjee J., Hsiao D. K., Baum R. I., Concepts and Capabilities of a Database Computer, *ACM Transactions on Database Systems*, 3(4), 347—384 (1978).
8. Kannan K., The Design of a Mass Memory for a Database Computer, Proceedings of the Fifth Annual Symposium on Computer Architecture, New York, ACM, 1978, pp. 44—51.

---

## ГЛАВА 22

### АРХИТЕКТУРА МАШИН, УПРАВЛЯЕМЫХ ПОТОКОМ ДАННЫХ<sup>1)</sup>

Рассмотренные выше варианты построения архитектуры вычислительных машин значительно отличаются от традиционной современной архитектуры и в то же время сохраняют ее некоторые общие основные характеристики. К таким характеристикам относятся, во-первых, последовательное увеличение содержимого счетчика команд при переходе от одной команды к другой; во-вторых, пассивная природа памяти (исключением является память ассоциативного типа); в-третьих, возможность параллельного выполнения операций, что подразумевает коммутацию процессора между различными процессами и принудительное разделение программы и данных таким образом, чтобы они обрабатывались одновременно разными процессорами.

Машинам потоков данных все перечисленные характеристики не свойственны: отсутствует понятие о последовательности выполняемых команд и передаче управления, память не рассматривается как пассивное хранилище переменных программы, и, наконец, вводятся средства для обнаружения и эффективного использования ситуаций, допускающих параллельную обработку в программе без явных на то указаний программиста.

Появление машин потоков данных обусловлено тремя основными причинами: потребностью в существенном увеличении вычислительной мощности, серьезными недостатками принципов построения современных языков программирования и наличием «узких мест» в физической структуре традиционных машин.

Первая причина — отсутствие достаточной вычислительной мощности — является значительным препятствием для примене-

---

<sup>1)</sup> Следует обратить внимание на то, что машины, управляемые потоком данных (машины потоков данных), разрабатываются совершенно для других целей, чем рассмотренные в предыдущих главах машины базы данных. Впрочем, у машин того и другого типа имеется ряд схожих свойств: высокая степень параллелизма выполнения операций и регулярности (однородности) структуры машины, наличие в памяти встроенных схем обработки информации.

ния машин при решении важных проблем. Хотя в наши дни ЭВМ с успехом применяют для экономических расчетов (подготовка платежных ведомостей, начисление страховых премий, бухгалтерский учет, ведение расчетов по кредитным карточкам), вычислительные машины плохо приспособлены для решения задач в области создания искусственного интеллекта, распознавания образов, обработки речевой информации, прогнозирования погоды, проектирования аэродинамических конструкций, перевода с одного иностранного языка на другой, сейсмического анализа и многих других. В качестве примеров, подтверждающих значительность проблемы, приведем следующие факты. Новому поколению машин автоматического перевода с одного иностранного языка на другой поиадобятся процессоры, способные выполнять 2—3 млрд. операций в секунду. Для составления местного краткосрочного прогноза погоды (на сутки) по результатам наблюдений за состоянием атмосферы потребуется скорость порядка 100 млрд. операций в секунду. Моделирование действия подъемной силы на крыло самолета и турбулентности воздушного потока возможно при наличии вычислительных систем, обеспечивающих выполнение 1 млрд. операций в секунду над числами с плавающей точкой. Эффективность будущих военных разработок зависит от бортовых вычислительных машин самолетов, кораблей и космических аппаратов, производительность которых должна быть по крайней мере на порядок выше, чем у современных аналогичных машин.

Традиционным решением задачи увеличения мощности системы (после того как производительность единственного процессора доведена до максимального значения) является использование нескольких процессоров. Однако такой подход не является удовлетворительным в силу действия следующих обстоятельств. Во-первых, возникают проблемы программирования, обусловленные необходимостью для программиста «подгонять» структуру данных и программ под жесткую структуру много-процессорной или распределенной вычислительной системы. Например, при работе с системой, содержащей 16 процессоров, программисту приходится разбивать свою программу на 16 или более параллельных процессов. Задача далеко не тривиальная, и современные языки программирования, средства обслуживания и отладки едва ли могут здесь оказать какую-либо существенную помощь.

Во-вторых, в традиционных мультипроцессорных системах в режиме разделения памяти между процессорами возможны наложения обращений к памяти — так называемая *интерференция обращений к памяти*. Поскольку каждый отдельный процессор стремится захватить определенную часть памяти, подключение новых процессоров к памяти, уже разделяемой

другими процессорами, может дать только ограниченную выгоду. Интерференция обращений к памяти может быть уменьшена, если каждый процессор снабдить локальным кэш-буфером (кэш-памятью). Однако при этом возникает другая проблема: некоторая ячейка памяти может оказаться привязанной к нескольким кэш-буферам, т. е. операция записи одного процессора может повлечь за собой искажение информации в кэш-буферах других процессоров. Таким образом, хотя мультипроцессорные системы имеют определенные достоинства, повышая степень готовности системы для решения различных задач, они все же уступают однопроцессорным аналогам по такому критерию, как отношение стоимости к производительности.

Второй причиной появления машин потоков данных являются недостатки, присущие самой природе современных языков программирования, и осознание того факта, что эти языки отражают в своей структуре основополагающие принципы архитектуры машин фон Неймана [1, 2]. Помимо перечисленных в гл. 2 принципов построения архитектуры к традициям, заложенным фон Нейманом, можно отнести также и внутреннюю организацию современных машин. Упомянутая организация предполагает использование пассивной памяти, процессора, выполняющего операции по изменению содержимого памяти, и устройства управления, воздействующего на процессор с помощью потока следующих одна за другой команд. Понятие «переменная» в языке программирования адекватно понятию «область пассивной памяти» в машине, понятие «передача управления» в языке (реализуемая, например, операторами GO TO, IF, DO, CALL) отображает устройство управления и счетчик команд в машине фон Неймана. Подобным же образом понятие «присваивание» является отображением определенной операции процессора (изменения содержимого области памяти машины). Использование потока данных для управления машиной позволило сделать выводы, с которыми согласились и специалисты в области теории языков: во-первых, три указанные в начале раздела основные характеристики традиционных машин являются искусственными, не соответствуют естественному порядку реализации алгоритмов решения задач и усложняют процесс программирования; во-вторых, современные языки программирования являются продуктом развития представлений не «снаружи внутрь» (т. е. не исходя из точки зрения программиста), а «изнутри наружу» (под сильным влиянием организации первых машин с запоминаемой программой).

Третьей причиной появления машин потоков данных является осознание того факта, что тракт, соединяющий процессор с памятью, является тем самым «узким местом», которое в основном и ограничивает эффективность системы (этот вывод

был сделан ранее в гл. 2). В современных вычислительных системах назначение программы заключается в изменении содержимого памяти, предоставляемой для данных этой программы. Достигается это путем «проталкивания» одиночных команд и данных через узкую магистраль, связывающую память и процессор.

### ПРИНЦИП ДЕЙСТВИЯ МАШИН ПОТОКОВ ДАННЫХ

Машины потоков данных производят наиболее сильное впечатление тем, что принципы их проектирования не базируются на основных свойствах и характеристиках традиционных машин и языков программирования. В архитектуре машин потоков данных отсутствует понятие «пассивная память данных», а в языке потоков данных нет понятия «переменная»: данные перемещаются из команды в команду по мере выполнения программы.

Кроме того, в данном случае не используются понятия «передача управления», «счетчик команд» и «светвление вычислительного процесса». Вместо этого команды (операторы) управляются данными. Считается, что команда готова к выполнению (т. е. ее выполнение разрешено), если данные присутствуют в каждом из ее входных портов и отсутствуют в выходном порте. Выполнение команды приводит к исчезновению данных в ее входных портах и появлению результата в выходном порте. Программа представляет собой направлений граф, образованный соединенными между собой командами: выходной порт одной команды соединен с входным портом другой команды. Таким образом, порядок выполнения команды определяется ие счетчиком команд, а движением потока данных в командах.

Указанные принципы выполнения команд иллюстрирует рис. 22.1. Здесь окружности обозначают *команды*, стрелки — *линии связи* между командами, а зачеркнутые круги — *данные*.

Первые три команды показаны в состоянии запрета на выполнение. У первой команды нет входных данных, у второй — данные присутствуют не на всех входных линиях, а в третьей — имеются данные — предыдущий результат — на выходной линии. Четвертая команда имеет все, что необходимо для получения разрешения на выполнение, т. е. готова к выполнению. Выполнение команды приводит к исчезновению данных со входных линий и появлению результата на выходной линии.

На рис. 22.2 с помощью рассмотренных символических обозначений описывается решение квадратного уравнения. Возникает необходимость ввести два дополнительных понятия. Первым является *размножитель*, который представляет собой опе-

рацию с одним входом и несколькими выходами. Он готов к работе, когда на входной линии данные присутствуют, а выходные линии пусты. Его функция — распределять входные данные по всем выходным линиям. Размножитель обозначается небольшим зачерненным кругом. (Не путать с обозначением данных (рис. 22.1)! — Прим. перев.) Вторым новым понятием является бесконечный источник констант для команды. Например, после

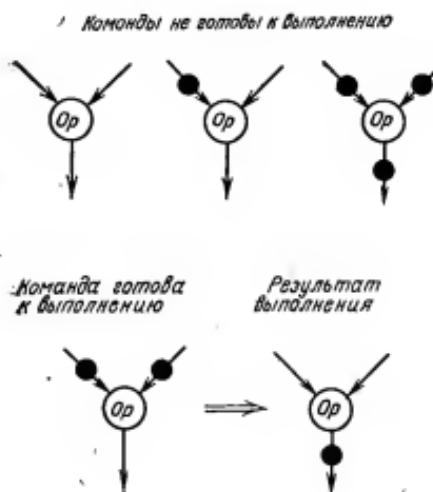


Рис. 22.1. Возможные состояния команд в машине потоков данных.

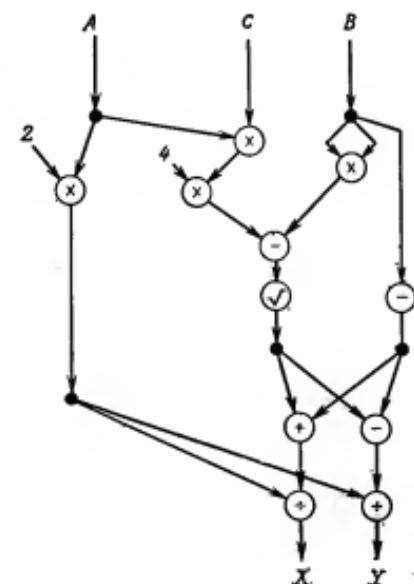


Рис. 22.2. Алгоритм решения квадратного уравнения в машине потоков данных.

выполнения операции умножения 2 на А, константа 2 немедленно снова воссоздается на одной из входных линий.

У читателя может возникнуть желание проследить путь перемещения данных по графу. (Отметим, что некоторые вопросы, такие, как порядок выполнения операций вычитания и деления, здесь не рассматриваются.) Следует обратить внимание на то, что целый ряд операций может оказаться готовым к выполнению в один и тот же момент времени и, следовательно, возможно их параллельное выполнение. Схема рис. 22.2 иллюстрирует тот принцип, что команды не запрашивают операнды традиционным способом, а указывают посредством соединительных линий, к каким командам направляются результаты их работы — их выходные линии. Таким образом, принципы построения гра-

фа потока данных во многом подобны принципам построения сетей Петри [3].

Граф потоков данных можно рассматривать и на более высоком уровне абстракции, или обобщения. Примером является схема, приведенная на рис. 22.3. Следовательно, и подлежащая решению задача может быть представлена с различной степенью обобщения, например такой, когда граф потоков данных полностью заменен единственным оператором на графике более высокого уровня.

### ЯЗЫК ПОТОКОВ ДАННЫХ

Прежде чем изучать архитектурные особенности машин потоков данных, рассмотрим более подробно сам принцип обслуживания этими машинами потоков данных на уровне языка программирования. С этой целью ознакомимся со специальным языком, предложенным Деннисом и его коллегами из Массачусетского технологического института [4—6], где проводятся наиболее серьезные исследования в области машин потоков данных.

Хотя язык Денниса является языком двумерного графического описания объектов программирования, существуют и другие предложения по построению подобных языков [7—11], предполагающие представление программ потоков данных в более привычном виде — в форме последовательности операторов, подчиняющихся определенному синтаксису такого языка.

Как упоминалось выше, в языке потоков данных не используются понятия «переменная» и «передача управления». Программа записывается в виде набора операторов, активируемых данными и соединенных односторонними линиями передачи. В основу языка Денниса положены следующие три основных понятия: исполнительный элемент, информация и линия связи.

*Исполнительный элемент* символизирует операцию, готовую к выполнению при поступлении информации на входные линии этого элемента и при отсутствии информации на его выходных линиях. Существуют два типа исполнительных элементов — блоки (*actor*) и размножители (*link*). *Блок* — это исполнительный элемент с одной выходной линией и одной или несколькими входными, *размножитель* — с одной входной линией и несколькими выходными.

*Информация* в языке Денниса представляется в виде токенов, которые передаются по линиям связи, обрабатываются и выдаются исполнительными элементами.

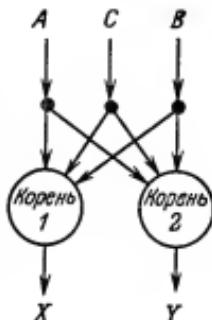


Рис. 22.3.  
Обобщенное представление алгоритма решения квадратного уравнения.

Различают два основных вида информации: **значения данных** (например, числовые величины) и **значения управляющих сигналов** (логические величины TRUE — ИСТИННО или FALSE — ЛОЖНО). В описываемом языке отсутствуют средства для распознавания типов значений данных (целые, с фиксированной точкой, с плавающей точкой, комплексные и т. д.). Некоторые расширения языка, связанные с обработкой структур данных, рассматриваются в одном из следующих разделов.

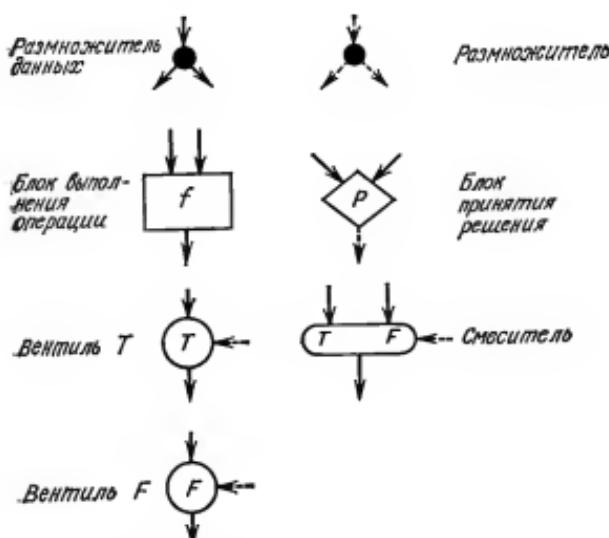


Рис. 22.4. Исполнительные элементы графического языка.

Используемое в языке Денинса понятие *линия связи* (arc) символизирует односторонний тракт, по которому информация передается от одного исполнительного элемента к другому. Сигналы на линии связи могут отсутствовать либо на ней может находиться только 1 токен информации. Следовательно, линию связи можно рассматривать как определенный эквивалент традиционных понятий «переменная» и «область памяти». В соответствии с классификацией информации на значения данных и значения управляющих сигналов линии также разделяются на *линии данных* (обозначены на рисунках сплошными стрелками) и *управляющие линии* (обозначены штриховыми стрелками).

На рис. 22.4 изображены исполнительные элементы языка: сплошные стрелки показывают места подсоединения линий данных, штриховые — управляющих линий. Операция размно-

жения готова к выполнению при появлении токена на единственной входной линии размножителя и при условии, что все его выходные линии пусты. Размножитель распределяет полученный токен по выходным линиям. Блок выполнения операции обычно имеет одну или две входные линии. Блок готов к работе при наличии токенов данных на всех его входных линиях и при условии, что выходная линия пуста. Он принимает входные токены, выполняет некоторые преобразования над полученными величинами и помещает результирующий токен данных на свою выходную линию. Типичными операциями таких блоков являются сложение, вычитание, умножение, инвертирование, извлечение квадратного корня и т. д.

Блок принятия решения функционирует аналогичным образом, однако результатом его работы является управляющий сигнал (логическая величина). В этом блоке вычисляется отношение, составленное из выходных данных, и формируется результат в виде логической величины TRUE (ИСТИННО) или FALSE (ЛОЖНО). Типичными операциями отношения являются операции сравнения двух величин по одному из критериев типа «равны», «не равны», «первая величина больше второй» и т. д.

Остальные три блока имеют на входе и данные, и управляющие сигналы. Блок типа вентиль T (T — от английского слова TRUE) готов к работе при наличии на его входах как токена данных, так и токена управляющей информации (при этом, как обычно, выходная линия должна быть пуста). Как и все остальные исполнительные элементы, этот блок «поглощает» входную информацию во время выполнения. Если значение управляющего сигнала TRUE, то имеющийся на входе блока токен данных передается на выходную линию. Если значение управляющего сигнала FALSE, то сигнал на выходе блока не формируется. Таким образом, вентиль T либо пропускает входные данные на свой выход, либо просто «поглощает» их. Блок типа вентиль F (F — от английского слова FALSE) работает аналогичным образом, только для передачи токена данных на выходную линию требуется управляющий сигнал, имеющий значение FALSE.

Среди различных блоков рассматриваемого языка потоков данных блок типа «смеситель» является исключением: выполнение предписываемых им действий не вызывает уничтожения всех токенов на входных линиях, и для перехода его в состояние готовности к работе не требуется наличия всех входных токенов. Этот блок готов к работе при соблюдении одного из следующих условий: 1) одновременное присутствие управляющего токена TRUE и токена данных на линии с меткой T; 2) одновременное присутствие управляющего токена FALSE и

токена данных на линии с меткой F. В обоих случаях выходная линия должна быть пуста. Если присутствует управляющий токен TRUE, токен данных с входной линии T пересыпается на выходную линию. В результате эти два токена на входных линиях уничтожаются, но при наличии токена данных на линии F он сохраняется. Если присутствует управляющий токен FALSE, то токен данных на входной линии F пересыпается на выходную линию. В результате этого два токена на входных линиях уничтожаются, но токен данных на линии T, если он присутствовал, сохраняется.

Свойства рассматриваемого языка нагляднее всего можно продемонстрировать на примере выполнения программы. Простая программа, выбранная для изучения, сначала записывается на языке типа ПЛ/I:

```
Z: PROCEDURE (X,Y);
DO I = 1 TO X;
  IF Y>1 THEN Y = Y*Y;
  ELSE Y = Y+Y+2;
END;
END;
```

Представление этой программы на языке потоков данных показано на рис. 22.5. Такая форма представления выглядит более громоздко, чем приведенная выше процедурная запись. Однако следует иметь в виду, что язык потоков данных еще не сформулирован окончательно и вполне возможно, что со вре-

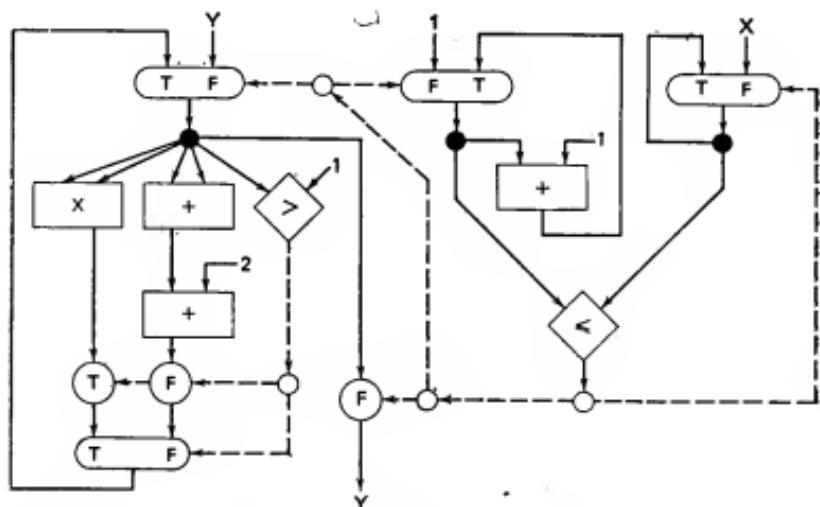


Рис. 22.5. Графическое представление программы потоков данных.

менем будет разработан синтаксис этого языка, более традиционный по форме.

Сопоставляя рис. 22.5 с программой на языке, подобном языку ПЛ/1, можно установить их взаимное соответствие. Левая часть рис. 22.5 представляет собой в основном тело цикла DO; правая часть отображает выполнение оператора DO. Отметим, что некоторые блоки имеют на входных линиях константы. Смысл этого заключается в том, что как только блок

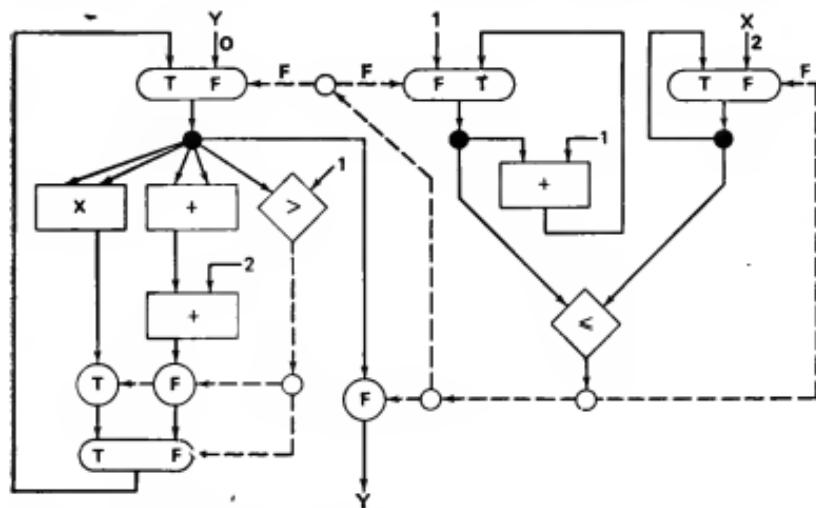


Рис. 22.6. Начальное состояние программы.

выполняет предписанную ему работу, константа вновь автоматически воспроизводится на линии.

Рассмотрим выполнение программы на языке потоков данных при значениях входных переменных  $X$  и  $Y$ , равных 2 и 0 соответственно. Прежде всего программа должна быть приведена в некоторое начальное состояние путем задания на графе определенных управляемых токенов. На рис. 22.6 показано такое начальное состояние программы. Управляемые токены на управляемых входных линиях трех смесителей в верхней части схемы имеют начальные значения. Управляемые токены и токены данных обозначаются на рисунках явным указанием их значений.

В соответствии с начальным состоянием программы, отображенным на рис. 22.6, три блока готовы к работе. Смеситель, расположенный вверху слева, готов к выполнению своих функций, поскольку к нему приложен управляющий токен FALSE, а на его входной линии F имеется токен данных. Два других смесителя находятся в состоянии готовности по тем же причинам.

иам. Таким образом, при наличии соответствующих аппаратных средств, реализующих архитектуру машины потока данных, эти три смесителя могут функционировать параллельно. Для обсуждения здесь хода выполнения программы следует рассматривать работу каждого блока шаг за шагом, принимая во внимание, что, во-первых, в каждый момент времени несколько блоков могут быть одновременно готовы к работе, т. е. обеспечена возможность их параллельной работы, и, во-вторых, ход выполнения программы не зависит от того, в каком порядке

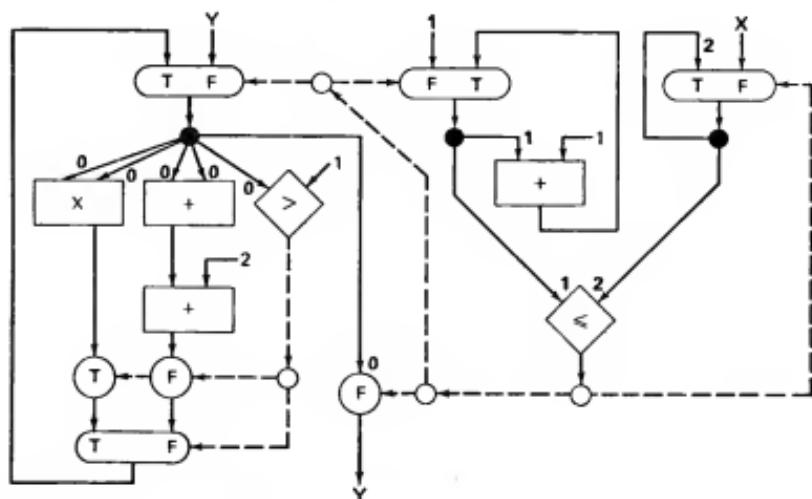


Рис. 22.7. Состояние программы в момент А.

функционируют готовые к работе блоки (если вообще существует какой-либо определенный порядок их функционирования).

Далее следует пропустить эти значения входных величин через программу, пока не будет получено значение результата на выходной линии. Поскольку пошаговый анализ этой процедуры займет много времени и весьма утомителен, ограничимся рассмотрением нескольких «мгновенных снимков» состояния программы в процессе выполнения указанных действий.

Результатом работы смесителя, находящегося вверху слева, является передача величины 0 через размножитель на шесть входных линий других блоков. В результате работы смесителя, расположенного справа, величина 2 через размножитель поступает обратно на вход Т того же смесителя и, кроме того, на блок принятия решения. Работа среднего смесителя завершается подачей величины 1 на исполнительный блок и блок принятия решения.

Полученное в результате описаний действий состояние программы — первый «мгновенный снимок» — показано на рис. 22.7. Теперь пять блоков готовы к работе и могут функционировать параллельно. Блок принятия решения по критерию «первый операнд отношения больше второго» формирует управляющий токен FALSE, который посыпается в три блока: вентили T и F и инжиниер смеситель. В результате срабатывания блока принятия решения по критерию «первый операнд отношения меньше или равен второму» формируется управляющий токен TRUE,

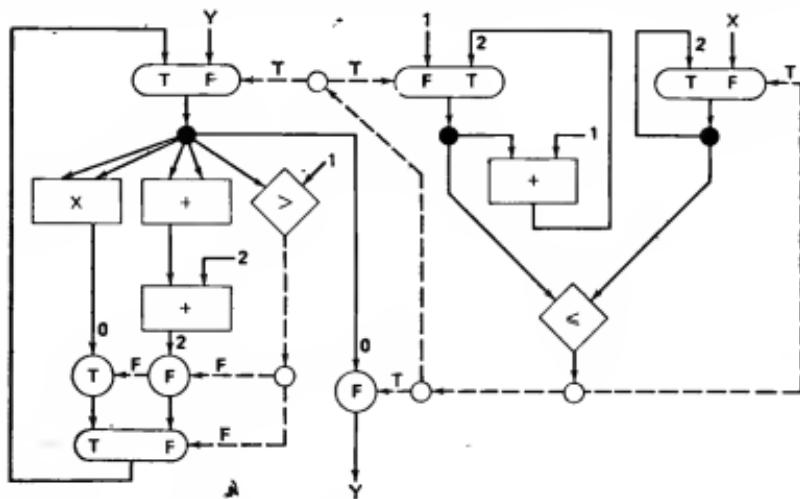


Рис. 22.8. Состояние программы в момент Б.

направляемый четырем адресатам. Блок операции сложения, расположенный в середине схемы, также готов к работе, он пошлет величину 2 обратно на вход среднего смесителя. Эта операция сложения эквивалентна сложению внутри цикла DO в программе на языке, подобном языку ПЛ/1.

На рис. 22.8 приведен второй «мгновенный снимок», отображающий следующее состояние программы. В этот момент готовы к работе три вентиля. Поскольку вентиль F, являющийся источником данных для выходной линии программы, имеет управляющий токен TRUE, величина 0 будет просто «поглощена» этим вентилем, и таким образом будет предотвращен вывод из программы ошибочного результата. То же самое произойдет и в вентиле T, но второй вентиль F формирует величину 2, которая переведет в состояние готовности расположенный под этим вентилем смеситель. Указанный смеситель в свою очередь перешел величину 2 на смеситель, находящийся вверху слева. Обратите внимание на то, как работает «арифметический раздел»

программы. Отказавшись от описания программы как совокупности процедур и изобразив ее графически, мы сумели отобразить тот факт, что оба арифметических выражения, определенных в операторе IF, могут вычисляться параллельно с вычислением значения отношения оператора IF ( $Y>1$ ). Описываемые действия завершаются работой вентилей T и F, осуществляющих выбор нужного вычисленного выражения с целью его передачи в качестве исходной величины для нового шага итерации.

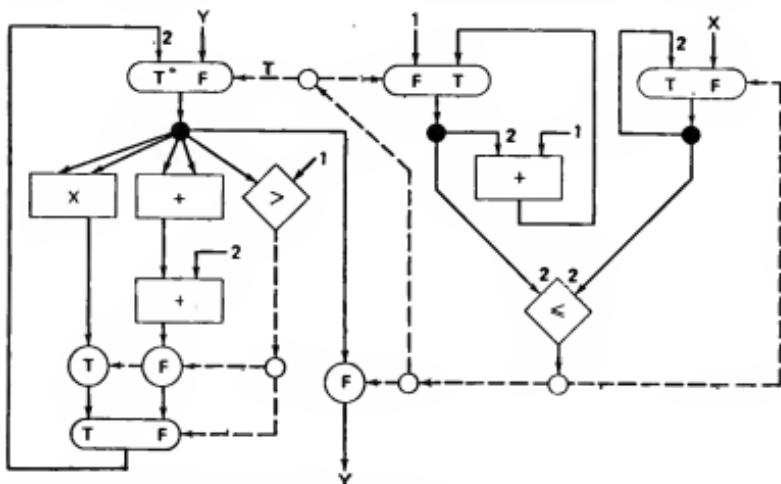


Рис. 22.9. Состояние программы в момент В.

Рис. 22.9 показывает состояние программы в один из последующих моментов (третий «мгновенный снимок»). Величина 2 будет возвращена в результате нового повторения циклического процесса в левой части графа. Таким образом, действие программы будет аналогично предыдущему, только изменятся значения токенов данных. Во время этого выполнения упомянутого циклического процесса на выходную линию программы никакие данные помещены не будут. Однако при этом блок принятия решения по критерию «первый операнд отношения больше второго» формирует на выходе значение TRUE, что дает возможность входным данным вентиля T пройти на его выход и далее, в верхнюю часть графа.

Рис. 22.10 иллюстрирует более позднее состояние программы (четвертый «мгновенный снимок»). В этот момент на выходе блока принятия решения по критерию «первый операнд отношения меньше или равен второму» формируется управляющий токен FALSE, рассылаемый четырем адресатам (точкам) программы. На рис. 22.11 показано соответствующее состояние

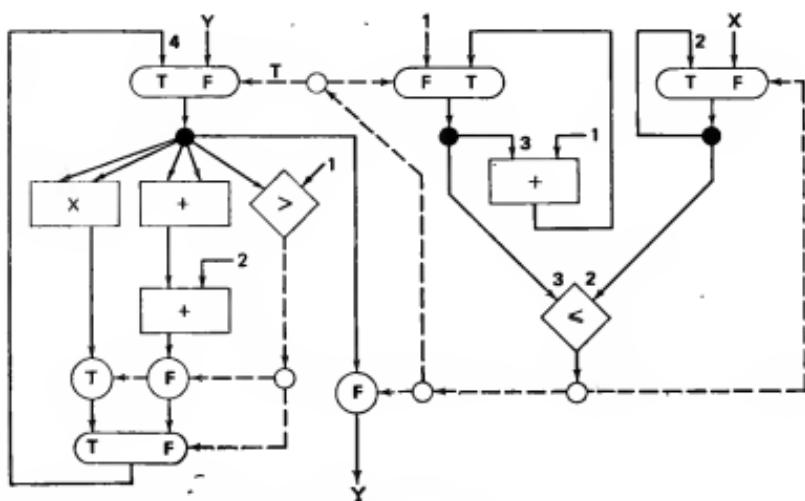


Рис. 22.10. Состояние программы в момент Г.

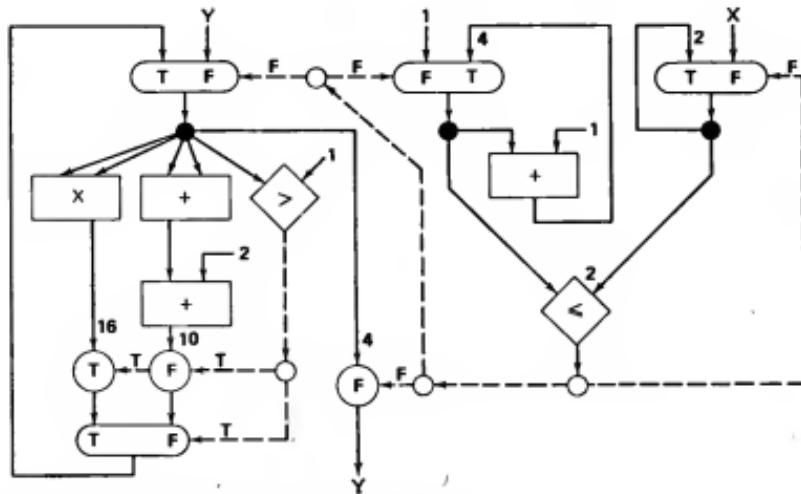


Рис. 22.11. Состояние программы в момент Д.

программы в этот момент времени: на входах вентиля F, расположенного в середине графа, сейчас имеются соответственно управляющий токен FALSE и токен данных, значение которого равно 4; в результате на выходной линии программы появится величина 4. Немаловажно также проследить продвижение потока остальных токенов по графу программы, поскольку программы потоков данных имеют одно любопытное свойство: их блоки могут все еще пребывать в состоянии готовности к выполнению.

нению соответствующих функций даже после того, как программа сформировала требуемый результат. Вентиль F, расположенный в левой нижней части, уничтожит оба своих входных сигнала, вентиль T пропустит величину 16, которая пройдет через смеситель в верхнюю часть графа, где дальнейшее продвижение этой величины будет заблокировано, поскольку на вход соответствующего смесителя подается управляющий токен ЛОЖНО. Таким же образом окажутся заблокированными величины 2 и 4, поступающие соответственно на правый и средний смесители.

Следовательно, вскоре после формирования на выходе результата программа прекращает функционирование, хотя и не произошло ее восстановление в нужное начальное состояние, показанное на рис. 22.6. (Граф содержит лишние токены данных, которых не было вначале.) Это является недостатком программы, поскольку упомянутые «застрявшие» токены будут препятствовать последующим попыткам выполнения этой программы. Можно было бы сделать так, чтобы программа сама уничтожала такие токены, но для упрощения проводимых рассуждений эти вопросы здесь не рассматриваются.

Подведем итог тому, что дает такое представление программы. Оно позволяет достичь высокой степени параллелизма выполнения операций, хотя с точки зрения программиста это не является основной целью программирования. Даже в такой весьма примитивной программе в каждый момент времени находилось несколько блоков, одновременно готовых к работе. Если предположить реализуемость подобного параллелизма в архитектуре вычислительной машины, то можно утверждать, что достижима высокая степень распараллеливания вычислений даже в небольших программах. Основа такой архитектуры рассматривается в следующем разделе.

## МАШИНА ПОТОКОВ ДАННЫХ

Перейдем теперь к рассмотрению структуры и архитектуры типичной машины потоков данных. Такая машина проектируется в Массачусетском технологическом институте [4—6, 12—18]. Это наиболее известный проект машины, однако уже был высказан ряд других предложений [8, 18—27].

Структура этой машины показана на рис. 22.12. Здесь нет ни процессора, ни памяти в их традиционном смысле. Машина разделена на три главные секции. Первой из них является память, содержащая ячейки команд. Как будет видно из дальнейшего рассмотрения, ячейка команды состоит из кода операции, одного или более входных портов и указателя ячеек команд, в которые должен быть направлен результат выполнения данной

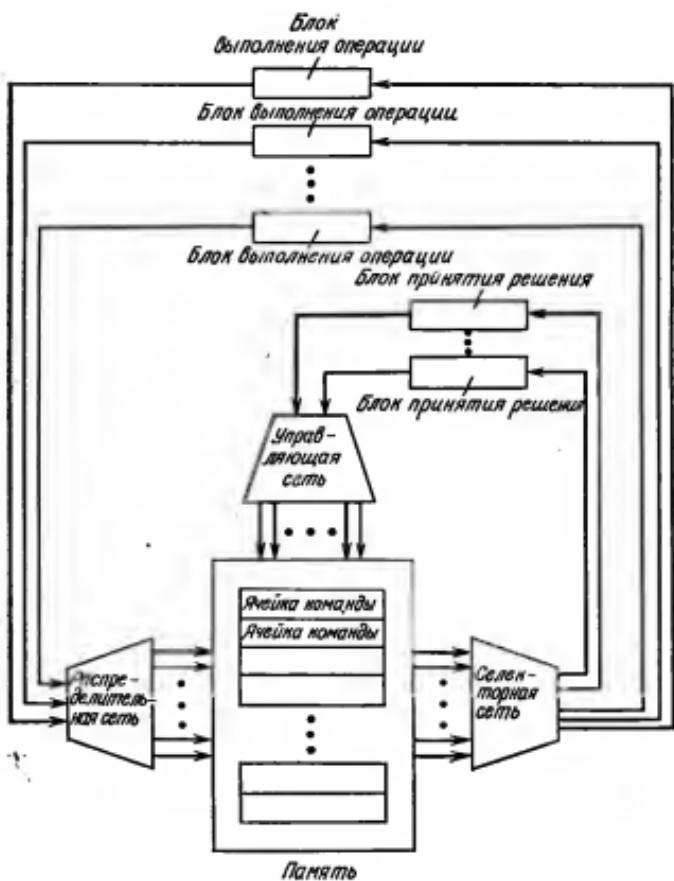


Рис. 22.12. Машина потоков данных, проектируемая в Массачусетском технологическом институте.

команды. Ячейки команд не являются пассивными запоминающими устройствами, они содержат некоторые логические схемы.

Вторая секция машины включает блоки выполнения операций и принятия решений. Эти блоки представляют собой устройства, выполняющие команды. Разница между ними заключается в том, что результатом работы блока выполнения операции являются данные (полученные, например, при реализации операций сложения или умножения), а результатом работы блока принятия решения — управляющая информация (логическая величина). Как показано, машина потоков данных содержит произвольное, насколько возможно большое количество указанных блоков. Поскольку их функции заключаются только в том, чтобы выполнить операцию, условия реализации которой опреде-

ляются так называемой *селекторной сетью*, и послать результат в другую сеть, блоки выполнения операций и принятия решений значительно проще традиционных процессоров.

Третья секция состоит из трех *переключающих сетей*. Селекторная сеть извлекает команду, готовую к выполнению, формирует так называемый командный пакет и направляет его к блоку выполнения операции или принятия решения. *Распределительная сеть* принимает пакет результата (данные и адрес назначения) и направляет данные в указанную ячейку команды. Точно так же *управляющая сеть* передает управляющий пакет (результат принятия решения и адрес назначения) в указанную ячейку команды.

Код операции	Адрес(а) назначения		
Вентиль-ный код	Вен-тильный флаг	Флаг дан-ных	Данные
Вентиль-ный код	Вен-тильный флаг	Флаг дан-ных	Данные
Вентиль-ный код	Вен-тильный флаг	Флаг дан-ных	Данные

Рис. 22.13. Содержимое ячейки команды.

МашинаРаботает следующим образом. Если ячейка команды располагает всеми необходимыми входными токенами, она посыпает командный пакет в селекторную сеть. Последняя направляет пакет к имеющемуся в распоряжении блоку выполнения операции или принятия решения. Порядок перехода команд в состояние готовности отличается от порядка, принятого в языке потоков данных. Дело в том, что в машине для команды единственным необходимым условием готовности к выполнению является наличие соответствующей входной информации; команда может выполняться, даже если занято место, куда должен поступать формируемый ею результат. В ходе выполнения командного пакета создаются один или несколько пакетов результата (по одному для каждого адресата), которые пересыпаются в распределительную или управляющую сеть. Если ячейка назначения пуста, результат переносится в нее. В противном случае он задерживается в сети до освобождения ячейки назначения. Таким образом, появляется вероятность перегрузки распределительной и управляющей сетей пакетами результатов, что может привести к неразрешимым — «тупиковым» — ситуациям. Оставим на некоторое время в стороне эту проблему и вернемся к ее анализу в одном из последующих разделов.

Программы потоков данных размещаются в ячейках команд. Операция, заданная в команде, соответствует одному из рас-

смотренных выше блоков, однако вентили Т и F, а также смеcитель и размножитель не имеют явно выраженных эквивалентов в наборе команд: их функции могут быть включены во все ячейки команд.

На рис. 22.13 в упрощенном виде представлено размещение команды в ячейке. Командой может быть, например, команда сложения, состояние готовности которой определяется наличием двух входных токенов.

Содержимое поля «вентильный код», задаваемое для каждого входа, определяет вентильные свойства, управляющие приемом данных. Ниже приведены четыре возможных значения этого кода:

N — вентильные функции не выполняются; любой результат, направляемый к данному входу, помещается в поле данных (разумеется, при условии, что оно свободно);

T — данные, направляемые в это поле, будут приняты, если управляющий токен TRUE (вентильный флаг) уже поступил или поступит в требуемый момент; при значении вентильного флага FALSE поступающие данные игнорируются вентилем;

F — данные, направляемые в это поле, будут приняты, если вентильный флаг, имеющий значение FALSE, уже поступил или поступит в требуемый момент времени; при значении вентильного флага TRUE поступающие на вентиль данные игнорируются;

C — содержимое поля является константой и не стирается в результате выполнения команды.

Вентильный флаг поступает из управляющей сети и может принимать одно из трех значений:

Off — управляющий пакет (вентильный флаг) не был получен;

T — получен управляющий пакет TRUE;

F — получен управляющий пакет FALSE.

Данные поступают из распределительной сети. Флаг данных указывает, содержит ли порт данные в текущий момент.

На рис. 22.14 приведено несколько ячеек команд в различных состояниях. Ячейка на рис. 22.14, а содержит команду сложения (Add), результат выполнения которой должен быть направлен в командные ячейки 8а и 22б. Обозначение 8а соответствует первому входному порту, или приемнику, ячейки 8, обозначение 22б — второму входному порту ячейки 22. При выполнении этой команды сложения произойдет прибавление константы 1 к приемаемому значению.

Ячейка на рис. 22.14, б содержит ту же команду в состоянии готовности к выполнению, когда из какой-то другой ячейки поступило значение 2. Звездочкой, расположенной рядом с ячейкой, будем отмечать готовые к выполнению команды. После того как командный пакет поступает в селекторную сеть, эта

Add	Ba 22b
N	
C	1

*а*

Add	Ba 22b
N	✓
C	1

*б*

Add	6b
T	
C	1

*в*

Add	6b
T	T
C	

*г*

Add	6b
T	✓
C	

*д*

Add	6b
T	T
C	

*е*

Add	6b
T	F
C	

*ж*

Equal	Ba 4b
N	
N	

*з*

Рис. 22.14. Ячейки команд в различных состояниях.

ячейка команды восстанавливает свое исходное состояние (рис. 22.14, а).

На рис. 22.14, в показана другая команда сложения. Она направляет результат своей работы во второй входной порт ячейки 6. На один из входов поступает константа, на другой — вентильный код Т. На рис. 22.14, г изображено такое состояние ячейки команды, когда управляющий пакет TRUE уже получен из другой команды, однако команда еще не готова к выполнению, потому что поле данных пусто. На рис. 22.14, д показано другое состояние: ячейка команды получила данные, но еще не готова к выполнению, так как не поступил управляющий сигнал TRUE. На рис. 22.14, е команда уже готова к выполнению. После выполнения состояние ячейки восстановится в соответствии с рис. 22.14, в.

На рис. 22.14, ж изображено другое возможное состояние: ячейка получила управляющий сигнал FALSE. Она останется в этом состоянии, пока не поступят данные. В этот момент и значение данных, и вентильный флаг F уничтожаются, поскольку коду вентиля присвоено значение Т. Без выполнения коман-

ды ячейка вернется в состояние, изображенное на рис. 22.14, в.

На рис. 22.14, з показана ячейка команды, формирующей управляющий сигнал. Она принимает два числа и после этого переходит в состояние готовности к выполнению. Эта команда выполняет проверку отношения указанных чисел на равенство и в качестве результата формирует управляющий пакет, содержащий логическую величину TRUE или FALSE. Этот пакет будет направлен в позиции вентильных флагов первого входного порта команды 8 и второго входного порта команды 4.

Рассмотрим теперь работу всей программы. На рис. 22.15 изображена машинная программа, соответствующая программе на языке потоков данных, приведенной на рис. 22.5. Поле флага данных здесь не показано, поскольку в последующих рассуждениях отсутствие данных принято обозначать пустым полем данных. Каждая команда XFER передает указанным в ней адресам

1	XFER	2a2b3a3b5a6a
	N	

7	XFER	2a2b3a3b5a6a
	T	

2	MULT	7a
	T	
	N	

8	XFER	2a2b3a3b5a6a
	F	

3	ADD	4a
	T	
	N	

9	XFER	11b 9a
	T	

4	ADD	8a
	N	
	C	2

10	ADD	10a 11a
	T	
	C	1

5	>	7a 8a
	T	
	C	1

11	$\leq$	2a3a5a6a10a9a
	N	
	N	

6	XFER	
	F	

Рис. 22.15. Машинная реализация программы, представленной на рис. 22.5.

1	<table border="1"> <tr> <td>XFER</td><td>2a2b3a3b5a6a</td></tr> <tr> <td>N</td><td>O</td></tr> </table>	XFER	2a2b3a3b5a6a	N	O	*	7	<table border="1"> <tr> <td>XFER</td><td>2a2b3a3b5a6a</td></tr> <tr> <td>T</td><td></td></tr> </table>	XFER	2a2b3a3b5a6a	T					
XFER	2a2b3a3b5a6a															
N	O															
XFER	2a2b3a3b5a6a															
T																
2	<table border="1"> <tr> <td>MULT</td><td>7a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td>N</td><td></td></tr> </table>	MULT	7a	T		N			8	<table border="1"> <tr> <td>XFER</td><td>2a2b3a3b5a6a</td></tr> <tr> <td>F</td><td></td></tr> </table>	XFER	2a2b3a3b5a6a	F			
MULT	7a															
T																
N																
XFER	2a2b3a3b5a6a															
F																
3	<table border="1"> <tr> <td>ADD</td><td>4a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td>N</td><td></td></tr> </table>	ADD	4a	T		N			9	<table border="1"> <tr> <td>XFER</td><td>11b 9a</td></tr> <tr> <td>T</td><td>T</td></tr> <tr> <td></td><td>2</td></tr> </table>	XFER	11b 9a	T	T		2
ADD	4a															
T																
N																
XFER	11b 9a															
T	T															
	2															
4	<table border="1"> <tr> <td>ADD</td><td>8a</td></tr> <tr> <td>N</td><td></td></tr> <tr> <td>C</td><td>2</td></tr> </table>	ADD	8a	N		C	2		10	<table border="1"> <tr> <td>ADD</td><td>10a 11a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td>C</td><td>1</td></tr> </table>	ADD	10a 11a	T		C	1
ADD	8a															
N																
C	2															
ADD	10a 11a															
T																
C	1															
5	<table border="1"> <tr> <td>&gt;</td><td>7a 8a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td>C</td><td>1</td></tr> </table>	>	7a 8a	T		C	1		11	<table border="1"> <tr> <td>&lt;</td><td>2a3a5a6a10a9a</td></tr> <tr> <td>N</td><td></td></tr> <tr> <td>N</td><td>1</td></tr> </table>	<	2a3a5a6a10a9a	N		N	1
>	7a 8a															
T																
C	1															
<	2a3a5a6a10a9a															
N																
N	1															
6	<table border="1"> <tr> <td>XFER</td><td></td></tr> <tr> <td>F</td><td></td></tr> </table>	XFER		F												
XFER																
F																

Рис. 22.16. Начальное состояние программы.

там данные со своего единственного входа, т. е. эта команда выполняет функции размножителя.

Рис. 22.16 показывает программу в исходном состоянии. Входная величина  $Y=0$  находится в ячейке 1 (порт а), входная величина  $X=2$  вместе с вентильным флагом TRUE — в ячейке 9 (порт а), величина 1 — ячейках 10 и 11 (порт а). В рассматриваемый момент времени готовы к выполнению команды ячеек 1 и 9.

Команда ячейки 1 пересыпает величину 0 шести адресатам, а команда ячейки 9 направляет величину 2 двум адресатам: самой себе и в порт б ячейки 11. В результате выполнения этих команд данные в ячейке 1, а также данные и вентильный флаг в ячейке 9 исчезают. Новое состояние программы показано на рис. 22.17. Теперь только одна команда готова к выполнению, а именно команда ячейки 11. Некоторые команды (ячейки 2, 3, 5, 6, 9 и 10) имеют необходимые данные на входе, но ожида-

ют поступления одного или нескольких вентильных флагов, после получения которых станут готовыми к выполнению. Выполнение команды 11 вызовет появление вентильного флага TRUE, который будет доставлен шести адресатам. После этого программа окажется в состоянии, показанном на рис. 22.18; теперь шесть команд готовы к выполнению.

При выполнении команд 2, 3, 9 и 10 формируются пакеты результатов, подлежащие доставке в соответствующие ячейки. В результате выполнения команды 5 создается управляющий пакет TRUE, который должен быть доставлен двум адресатам. Команда 6 выполняться не будет, поскольку оказывается, что хотя она и имеет на входе данные и вентильный флаг TRUE, значение ее кода вентиля равно F, и, следовательно, указанные данные и вентильный флаг будут уничтожены. Полагая, что все упомянутые команды выполняются параллельно, приходим к заключению, что они «поглощают» свои входные данные, в

1	<table border="1"><tr><td>XFER</td><td>2a2b3a3b5a6a</td></tr><tr><td>N</td><td></td></tr></table>	XFER	2a2b3a3b5a6a	N		7	<table border="1"><tr><td>XFER</td><td>2a2b3a3b5a6a</td></tr><tr><td>T</td><td></td></tr></table>	XFER	2a2b3a3b5a6a	T					
XFER	2a2b3a3b5a6a														
N															
XFER	2a2b3a3b5a6a														
T															
2	<table border="1"><tr><td>MULT</td><td>7a</td></tr><tr><td>T</td><td></td></tr><tr><td>N</td><td>0</td></tr></table>	MULT	7a	T		N	0	8	<table border="1"><tr><td>XFER</td><td>2a2b3a3b5a6a</td></tr><tr><td>F</td><td></td></tr></table>	XFER	2a2b3a3b5a6a	F			
MULT	7a														
T															
N	0														
XFER	2a2b3a3b5a6a														
F															
3	<table border="1"><tr><td>ADD</td><td>4a</td></tr><tr><td>T</td><td></td></tr><tr><td>N</td><td>0</td></tr></table>	ADD	4a	T		N	0	9	<table border="1"><tr><td>XFER</td><td>11b 9a</td></tr><tr><td>T</td><td></td></tr><tr><td></td><td>2</td></tr></table>	XFER	11b 9a	T			2
ADD	4a														
T															
N	0														
XFER	11b 9a														
T															
	2														
4	<table border="1"><tr><td>ADD</td><td>8a</td></tr><tr><td>N</td><td></td></tr><tr><td>C</td><td>2</td></tr></table>	ADD	8a	N		C	2	10	<table border="1"><tr><td>ADD</td><td>10a 11a</td></tr><tr><td>T</td><td></td></tr><tr><td>C</td><td>1</td></tr></table>	ADD	10a 11a	T		C	1
ADD	8a														
N															
C	2														
ADD	10a 11a														
T															
C	1														
5	<table border="1"><tr><td>&gt;</td><td>7a 8a</td></tr><tr><td>T</td><td></td></tr><tr><td>C</td><td>1</td></tr></table>	>	7a 8a	T		C	1	11	<table border="1"><tr><td>&lt;</td><td>2a3a5a6a10a9a</td></tr><tr><td>N</td><td></td></tr><tr><td>N</td><td>2</td></tr></table>	<	2a3a5a6a10a9a	N		N	2
>	7a 8a														
T															
C	1														
<	2a3a5a6a10a9a														
N															
N	2														
6	<table border="1"><tr><td>XFER</td><td></td></tr><tr><td>F</td><td>0</td></tr></table>	XFER		F	0										
XFER															
F	0														

Рис. 22.17. Состояние программы в момент А.

1	<table border="1"><tr><td>XFER</td><td>2a2b3a3b5a6a</td></tr><tr><td>N</td><td></td></tr></table>	XFER	2a2b3a3b5a6a	N		7	<table border="1"><tr><td>XFER</td><td>2a2b3a3b5a6a</td></tr><tr><td>T</td><td></td></tr></table>	XFER	2a2b3a3b5a6a	T									
XFER	2a2b3a3b5a6a																		
N																			
XFER	2a2b3a3b5a6a																		
T																			
2	<table border="1"><tr><td>MULT</td><td>7a</td></tr><tr><td>T</td><td>T</td><td>0</td></tr><tr><td>N</td><td></td><td>0</td></tr></table>	MULT	7a	T	T	0	N		0	8	<table border="1"><tr><td>XFER</td><td>2a2b3a3b5a6a</td></tr><tr><td>F</td><td></td></tr></table>	XFER	2a2b3a3b5a6a	F					
MULT	7a																		
T	T	0																	
N		0																	
XFER	2a2b3a3b5a6a																		
F																			
3	<table border="1"><tr><td>ADD</td><td>4a</td></tr><tr><td>T</td><td>T</td><td>0</td></tr><tr><td>N</td><td></td><td>0</td></tr></table>	ADD	4a	T	T	0	N		0	9	<table border="1"><tr><td>XFER</td><td>11b 9a</td></tr><tr><td>T</td><td>T</td><td>2</td></tr></table>	XFER	11b 9a	T	T	2			
ADD	4a																		
T	T	0																	
N		0																	
XFER	11b 9a																		
T	T	2																	
4	<table border="1"><tr><td>ADD</td><td>8a</td></tr><tr><td>N</td><td></td><td></td></tr><tr><td>C</td><td></td><td>2</td></tr></table>	ADD	8a	N			C		2	10	<table border="1"><tr><td>ADD</td><td>10a 11a</td></tr><tr><td>T</td><td>T</td><td>1</td></tr><tr><td>C</td><td></td><td>1</td></tr></table>	ADD	10a 11a	T	T	1	C		1
ADD	8a																		
N																			
C		2																	
ADD	10a 11a																		
T	T	1																	
C		1																	
5	<table border="1"><tr><td>&gt;</td><td>7a 8a</td></tr><tr><td>T</td><td>T</td><td>0</td></tr><tr><td>C</td><td></td><td>1</td></tr></table>	>	7a 8a	T	T	0	C		1	11	<table border="1"><tr><td>&lt;</td><td>2a3a5a6a10a9a</td></tr><tr><td>N</td><td></td><td></td></tr><tr><td>N</td><td></td><td></td></tr></table>	<	2a3a5a6a10a9a	N			N		
>	7a 8a																		
T	T	0																	
C		1																	
<	2a3a5a6a10a9a																		
N																			
N																			
6	<table border="1"><tr><td>XFER</td><td></td></tr><tr><td>F</td><td>T</td><td>0</td></tr></table>	XFER		F	T	0													
XFER																			
F	T	0																	

Рис. 22.18. Состояние программы в момент Б.

результате чего программа переходит в состояние, показанное на рис. 22.19. В этот момент готовы к выполнению три команды. Читатель, наверное, сможет самостоятельно проследить оставшуюся часть процесса выполнения программы — до того момента, когда команда ячейки 6 сформирует выходное значение и программа перейдет в состояние покоя.

### СЕТИ ТРАКТОВ ПЕРЕДАЧИ ПАКЕТОВ

Одной из основных проблем в традиционных мультипроцессорных системах является организация связей между запоминающими устройствами и процессорами. В машине потоков данных, рассмотренной в предыдущем разделе, эта проблема разрешается путем использования сопрягающих сетей, обладающих высокой степенью внутреннего параллелизма функционирования. Этот параллелизм достигается благодаря тому, что боль-

шое количество команд может передавать свои данные через сети одновременно.

Селекторная, распределительная и управляющая сети, показанные на рис. 22.12, представляют собой сети трактов передачи пакетов. Когда команда готова к выполнению, формируется пакет операций, посылаемый в селекторную сеть. В функции этой сети входят направлять пакеты операций из большого числа ячеек команд в меньшее число блоков выполнения операций и принятия решения, обеспечивая при этом поступление пакета к соответствующему блоку. Выполнение команды приводит к формированию одного или нескольких пакетов данных или управляющих пакетов (один пакет для каждого адресата). Эти пакеты проходят через распределительную и управляющую сети и направляются к указанным ячейкам команд.

Все три сети могут быть построены на основе элементов двух типов (рис. 22.20). Один из них — *селектор*, передающий пакет с одного из своих входов на выход. Селектор обслужи-

1	<table border="1"> <tr> <td>XFER</td><td>2a2b3a3b5a6a</td></tr> <tr> <td>N</td><td></td></tr> </table>	XFER	2a2b3a3b5a6a	N		7	<table border="1"> <tr> <td>XFER</td><td>2a2b3a3b5a6a</td></tr> <tr> <td>T</td><td>F</td></tr> <tr> <td></td><td>0</td></tr> </table>	XFER	2a2b3a3b5a6a	T	F		0		
XFER	2a2b3a3b5a6a														
N															
XFER	2a2b3a3b5a6a														
T	F														
	0														
2	<table border="1"> <tr> <td>MULT</td><td>7a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td>N</td><td></td></tr> </table>	MULT	7a	T		N		8	<table border="1"> <tr> <td>XFER</td><td>2a2b3a3b5a6a</td></tr> <tr> <td>F</td><td>F</td></tr> <tr> <td></td><td></td></tr> </table>	XFER	2a2b3a3b5a6a	F	F		
MULT	7a														
T															
N															
XFER	2a2b3a3b5a6a														
F	F														
3	<table border="1"> <tr> <td>ADD</td><td>4a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td>N</td><td></td></tr> </table>	ADD	4a	T		N		9	<table border="1"> <tr> <td>XFER</td><td>11b 9a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td></td><td>2</td></tr> </table>	XFER	11b 9a	T			2
ADD	4a														
T															
N															
XFER	11b 9a														
T															
	2														
4	<table border="1"> <tr> <td>ADD</td><td>8a</td></tr> <tr> <td>N</td><td></td></tr> <tr> <td>C</td><td>0</td></tr> </table>	ADD	8a	N		C	0	10	<table border="1"> <tr> <td>ADD</td><td>10a 11a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td>C</td><td>2</td></tr> </table>	ADD	10a 11a	T		C	2
ADD	8a														
N															
C	0														
ADD	10a 11a														
T															
C	2														
5	<table border="1"> <tr> <td>&gt;</td><td>7a 8a</td></tr> <tr> <td>T</td><td></td></tr> <tr> <td>C</td><td>1</td></tr> </table>	>	7a 8a	T		C	1	11	<table border="1"> <tr> <td>&lt;</td><td>2a3a5a6a10a9a</td></tr> <tr> <td>N</td><td></td></tr> <tr> <td>N</td><td>2</td></tr> </table>	<	2a3a5a6a10a9a	N		N	2
>	7a 8a														
T															
C	1														
<	2a3a5a6a10a9a														
N															
N	2														
6	<table border="1"> <tr> <td>XFER</td><td></td></tr> <tr> <td>F</td><td></td></tr> </table>	XFER		F											
XFER															
F															

Рис. 22.19. Состояние программы в момент В.

ет свои входы по очереди, так что каждый вход рано или поздно будет опрошен. Другой элемент — *переключатель* — направляет пакет со своего единственного входа на один из выходов, при этом выбор нужного выхода производится на основе некоторых характеристик пакета. В селекторной сети в качестве такой характеристики переключатель использует код операции, в распределительной или управляющей сети — адрес пункта назначения пакета. Как будет показано ниже, сети включают в себя элементы еще трех типов: *буферы* — для временного хранения информации, а также *преобразователи кодов* из последовательного в параллельный и из параллельного в последовательный.

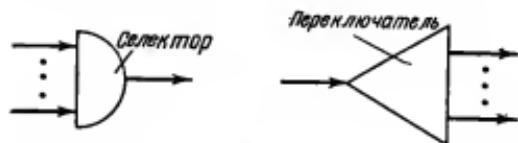


Рис. 22.20. Основные элементы сетей.

На рис. 22.21 показана функциональная схема простой селекторной сети. Эта небольшая сеть принимает операционные пакеты из 16 ячеек команд и направляет их к четырем процессорам. Селекторную сеть можно представить себе в виде черного ящика с большим количеством входов (по одному на каждую ячейку команды) и меньшим количеством выходов (по одному на процессор). Вследствие большого количества входов возникает проблема минимизации числа соединений между селекторной сетью и памятью команд. Поэтому указанная сеть принимает информацию из ячеек команд в виде последовательности битов. В то же время со стороны выходов желательно минимизировать время, в течение которого процессор занят выполнением команд, и, следовательно, целесообразно передавать пакет в процессор весь сразу. Таким образом, в процессе прохождения пакетов команд по селекторной сети осуществляется преобразование информации пакетов из последовательного кода в параллельный.

Параллелизм функционирования сети возможен благодаря использованию буферов, обеспечивающих временное хранение командных пакетов на входах каждого переключателя и селектора. Так, в небольшой селекторной сети, показанной на рис. 22.21, могло бы находиться до 30 пакетов команд. Селекторы и переключатели функционируют асинхронно, поэтому асинхронно и продвижение пакетов через сеть, за исключением случаев «конкуренции».

Управляющая и распределительная сети строятся на тех же элементах, что и селекториальная сеть, но отличаются от последней конфигурацией схемы соединения этих элементов (рис. 22.22). Распределительная и управляющая сети имеют малое количество входов и большое количество выходов. По причинам, рассмотренным выше, эти сети получают пакеты в параллельном коде, а затем по мере их прохождения выполняют преобразование параллельного кода в последовательный, в

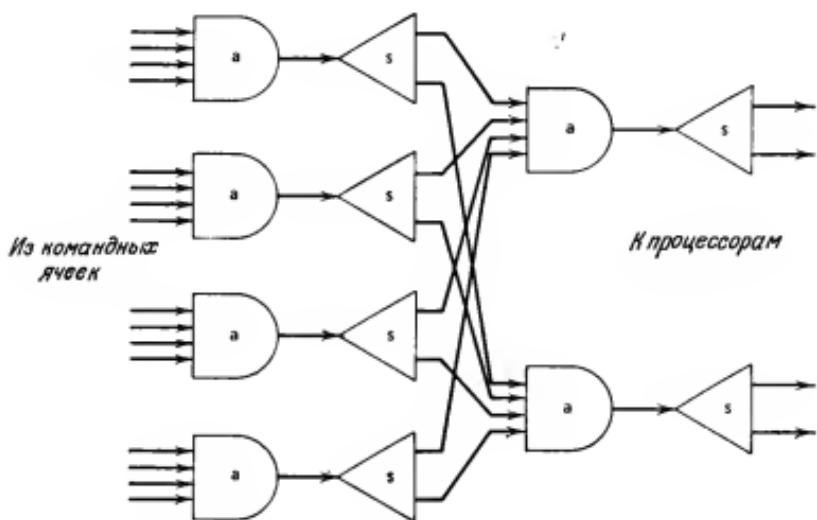


Рис. 22.21. Организация небольшой селекторной сети.

результате чего в ячейки команд пакеты поступают в виде последовательности битов.

Если проследить порядок прохождения пакета в схемах на рис. 22.21 или 22.22, то можно сделать заключение, что существует только один возможный путь из определенного входа в заданный выход. Однако в больших сетях возможно отступление от этого правила.

Анализ времени, необходимого для выполнения команды, показывает, что оно является суммой времени задержки пребывания команды в селекторной сети, времени выполнения команды в процессоре и времени задержки в распределительной или управляющей сети. Время работы процессора фиксировано, а задержки в сетях носят стохастический характер. Эти задержки являются функцией количества уровней в сети и числа других пакетов в этой сети. Поскольку число пакетов в сети в каждый момент времени зависит от особенностей конк-

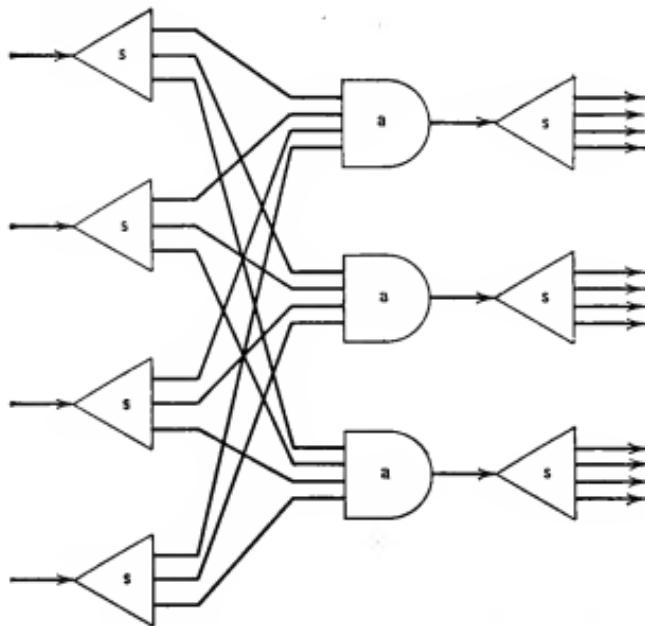


Рис. 22.22. Организация небольшой распределительной сети.

ретной программы, анализ производительности представляет собой непростую задачу.

Продвижение пакетов через сеть может увеличить суммарное время выполнения каждой отдельной команды (до десятков микросекунд), но это не обязательно приведет к заметному снижению производительности машины, поскольку в движении одновременно находится большое число пакетов. Длительность продвижения становится «узким местом» только в тех случаях, когда готовым к выполнению оказывается лишь небольшое число команд. (Предположим, программа достигла состояния, когда готова к выполнению только одна команда.) Например, если в машине 20 процессоров — блоков выполнения операций и принятия решения, — каждый из которых может обработать командный пакет в среднем за 200 ис, то в целом производительность машины может быть оценена быстродействием, равным 100 млн. операций в секунду. Такая скорость достижима даже в условиях чрезвычайно медленного прохождения отдельного пакета по сетям, если в любой момент времени найдется по меньшей мере 20 готовых к выполнению команд, если селекторная сеть может поставлять 20 командных пакетов каждые 200 ис и если распределительная и управляющая сети обладают примерно такими же возможностями. Последнее необходимо

мо для того, чтобы поддерживать максимальное число готовых к выполнению команд.

Из рассмотренного выше можно сделать интересный вывод, что сети трактов передачи пакетов моделируют основные принципы функционирования машин потоков данных на физическом уровне; сами сети могут рассматриваться как программы на языке потоков данных или как сети Петри. Таким образом, налицо два уровня параллелизма вычислений. Один уровень определяется наличием множества команд, одновременно готовых к выполнению и доступных для параллельной обработки многими процессорами. Другой, нижний уровень определяется прохождением этих команд и их результатов через сети, которые сами в большой степени обладают свойством параллелизма функционирования.

Еще одним свойством машин потоков данных, которое теперь становится очевидным, является высокая степень однородности логической структуры машины. Уже ячейка команды содержит значительную часть различных элементов, определяющих структуру машины в целом. Каждая ячейка включает запоминающее устройство и логические схемы проверки состояния готовности команды к выполнению. Другой формой проявления однородности логической структуры машины потоков данных является наличие большого количества блоков выполнения операций и принятия решений. И наконец, только что было показано, что сетям трактов передачи пакетов также присуща высокая степень однородности структуры, поскольку они состоят из большого числа селекторов, переключателей, буферов и преобразователей. Существуют проекты, которые идут еще дальше в осуществлении этого принципа и демонстрируют способ построения сети из маршрутных модулей  $2 \times 2$  вместо переключателей и селекторов [28, 32]. Подобная однородность логической структуры машин потоков данных делает принципы их организации особенно привлекательными при широком внедрении сверхбольших интегральных схем, применение которых в свою очередь расширяет возможности по выбору той или иной конфигурации машины.

## ПЕРЕПОЛНЕНИЕ И ТУПИКОВЫЕ СИТУАЦИИ

В машине потоков данных, описанной в предыдущем разделе, перевод команд в состояние готовности к выполнению производится независимо от того, доступны или нет их адресаты. Таким образом, команда может быть выполнена и один или несколько пакетов результата посланы в распределительную сеть до того, как команды-получатели будут способны принять их.

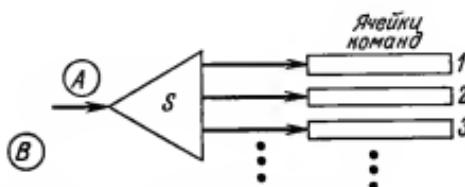


Рис. 22.23. Тупиковая ситуация в распределительной сети.

Первой проблемой, порождаемой такими принципами работы машины, может оказаться перегрузка распределительной и управляющей сетей. Пакет результата, находящийся в распределительной сети, размещается в буфере, связанным с соответствующим

селектором или переключателем. Этот пакет будет препятствовать продвижению последующих пакетов через данный участок сети до тех пор, пока находится там. Большое число пакетов, пребывающих в таком состоянии, может привести к блокировке движения в сети.

Второй проблемой является возможность возникновения тупиковых ситуаций. Тупиковая ситуация имеет место в том случае, если пакет результата А задерживает пакет результата В, в то время как пакет В необходим для перевода в состояние готовности той самой команды, для которой предназначен и к которой направлен пакет А. Такая ситуация показана на рис. 22.23. Допустим, что команда З не переводится в состояние готовности, поскольку данные поступили в порт Зб и не поступили в порт За. Допустим также, что некоторый пакет А находится в состоянии ожидания в переключателе, так как пунктом его назначения является порт Зб. Следовательно, пакет А будет задерживать движение всех остальных пакетов через этот переключатель. Где-то в распределительной сети находится пакет В, направляемый в порт За. Однако он никогда не сможет достичь пункта назначения. Создалась тупиковая ситуация.

Обе указанные проблемы решаются путем введения сигналов обратной связи между блоками выполнения операций [4, 29]. В дополнение к ранее определенным видам линий связи введем еще один вид — *сигнальные линии*. Сигнальная линия обозначается пунктиром и используется для передачи сигнала подтверждения от одного блока выполнения операций к другому, предыдущему. Этот сигнал вырабатывается блоком специального вида, называемым *сигнальным блоком* (рис. 22.24). Сигнальный блок переводится в состояние готовности, когда на его входную линию подается токен любого вида — либо токен данных, либо управляющий токен. Блок „поглощает“ такой токен; при этом формируется сигнал подтверждения на выходной линии блока. Описанные выше блоки должны быть модифицированы таким образом, чтобы сигнал обратной связи использовался в качестве дополнительного входного сигнала. Пример модификации функционального блока представлен на рис. 22.24.

Здесь показан блок с тремя входами: две обычные линии данных и сигнальная линия. Блок переводится в состояние готовности к выполнению, если токены данных поступили на входные линии данных, а сигнальный токен — на сигнальную линию. Блок «поглощает» входную информацию, выполняет необходимые преобразования данных и формирует результат в виде токена данных.

В некоторых случаях объектом функциональных преобразований может служить сам сигнал обратной связи. Блок OR

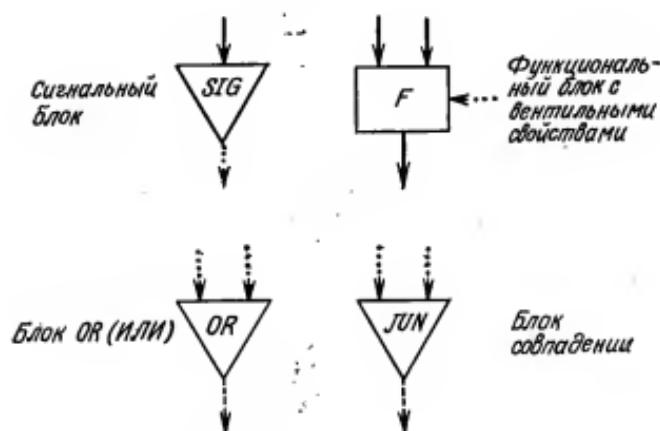


Рис. 22.24. Блоки, оперирующие сигналами подтверждения.

(ИЛИ), показанный на рис. 22.24, переводится в состояние готовности, когда упомянутый сигнал подается на любую из его входных линий. Сигнал поступает в блок и выдается на выходной линии. Блок совпадения (JUN) переводится в состояние готовности, когда сигнальные токены присутствуют на всех его входных линиях. Он «поглощает» эти сигналы и формирует сигнал на выходной линии.

Сигналы обратной связи применяют для предотвращения тупиковых ситуаций следующим образом. Каждый блок, который в принципе может выдать несколько токенов на свою выходную линию (т. е. может выполнить свою операцию неоднократно), преобразуют в подобный же блок вентильного типа, управляемый сигналом обратной связи с выхода следующего за ним блока (или блоков). Если результат операции, выполняемой данным блоком, посыпается в несколько пунктов назначения, для формирования общего сигнала подтверждения о доступности всех адресатов может использоваться блок совпадения.

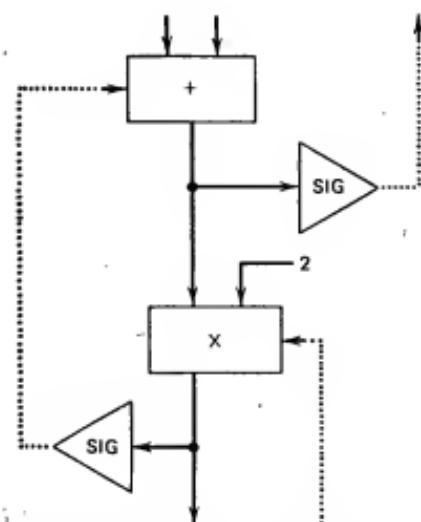


Рис. 22.25. Пример использования сигналов подтверждения.

На рис. 22.25 представлен простой пример такого решения. Здесь при выполнении операции сложения формируется сигнал подтверждения, который посыпается обратно в блок или блоки — источники входных данных для данного блока, реализующего операцию сложения. Блок операции умножения возвращает сигнал подтверждения блоку операции сложения. Блок, которому передается результат умножения, в свою очередь выдаст сигнал подтверждения.

Манипулирование сигналами обратной связи может быть предусмотрено в существующем наборе машинных команд без введения для этих

целей дополнительных команд. На рис. 22.26 показана команда ADD, по которой производится сложение переданного ей значения данных с константой 3. Команда претерпела следующие изменения. Во-первых, в поле, следующем за полем кода операции, указывается количество сигнальных пакетов, которые должны быть получены командой для ее перевода в состояние готовности к выполнению. Если это поле содержит 0, то команда не обладает вентильными свойствами и может быть приведена в состояние готовности к выполнению сразу же по получении токена данных. Если же в поле находится 1, то для перевода команды в состояние готовности ей необходимы и посыпка данных, и сигнал подтверждения. Команды, которые в этом поле содержат числа больше чем 1, обладают возможностями встроенной схемы совпадения. Команде ADD, показанной на рис. 22.26, для перевода в состояние готовности необходимо поступление двух сигнальных пакетов.

Во-вторых, еще одно дополнительное поле в команде отведено под указатель количества полученных к текущему моменту сигналов обратной связи. Когда значение содержимого этого поля становится равным 2 и поступает пакет данных, команда готова к выполнению. После «поглощения» пакета данных содержимому этого поля присваивается нулевое значение.

В-третьих, модифицировано указание адресатов команды. Теперь адреса, по которым посыпается результат выполнения команды, помечаются символическими метками D и S, как по-

казано на рис. 22.26. Пакеты результатов выполнения этой команды будут направлены в ячейки команд 8а и 9б, а сигнал подтверждения — в ячейку 4.

Формирование сигнальных пакетов может быть реализовано достаточно просто. Для их генерирования не требуется ожидать завершения выполнения команды. Сигнал может быть сформирован, как только команда становится готовой к выполнению и извлекается селекторной сетью, поскольку с этого момента ячейка команды доступна для новых входных данных. Следовательно, указанная сеть может прежде всего проверить, должен

ADD	2		D - 8а, 9б S - 4
N			
C			3

Рис. 22.26. Ячейка команды, рассчитанная на операции с сигналами подтверждения.

ли данный командный пакет в результате выполнения выдавать сигнальный пакет (или пакеты). При необходимости сигнальный пакет может быть сразу сформирован селекторной сетью и направлен в управляющую сеть для пересылки соответствующему адресату.

## ОБРАБОТКА СТРУКТУР ДАННЫХ

До сих пор за пределами рассмотрения оставался вопрос о том, каким образом программы потоков данных оперируют совокупностями данных, такими, как массивы или структуры. Очевидно, что использование для этой цели многозначных токенов неудобно, если массивы или структуры имеют значительные размеры. Перемещение длинных векторов сквозь селекторную и распределительную сети во время обработки является весьма нежелательным.

Один из вариантов решения этой проблемы предусматривает, во-первых, введение в машину потоков данных дополнительной памяти, предназначенией для хранения не команд, а структур данных, и во-вторых, создание средств адресации этой памяти. Прежде чем перейти к вопросу физической реализации указанного подхода, рассмотрим типы структур данных и свя-

занные с иими блоки [5, 30, 31]. В языке потоков данных определены три типа данных:

«Пусто»	Отсутствие данных
Элементарные данные	Данные скалярного типа или управляющая информация
Структура	Упорядоченная совокупность данных любого из трех типов

Упрощению структуру можно изобразить в виде двоичного дерева, в котором каждый элемент — узел — представляет данные типа «пусто», элементарные данные или структуру. На рис. 22.27 представлена структура по имени S, размещенияя в пассивной памяти данных.

Для адресации к любому элементу структуры необходимо

указать ее имя и двоичную последовательность переменной длины, которая определяет маршрут поиска на двоичном дереве (0 соответствует левой ветви, а 1 — правой). Применительно к дереву, показанному на рис. 22.27, адрес S,1 ссылается на узел, содержащий величину 6,2. Адрес S,000 ссылается на величину 3,6. Адрес S,00 определяет двухэлементную подструктуру, содержащую величины 3,6 и 7.

Для выполнения операций над структурами вводятся четыре новых блока. Первый из них — блок выбора. Он имеет две входные линии: для имени (адреса) структуры и для двоичной строки.

В результате работы блока выбора определяется значение элемента (узла) структуры, соответствующего указанному имени структуры и заданному местоположению этого элемента. Если в узле содержатся элементарные данные, то на выходной линии блока появляется значение этих данных. Если же узел является структурой (именуемой в этом случае подструктурой), то блок выбора выдает на выходную линию имя — адрес подструктуры. При подаче на входы блока выбора значения S и величины 010 (см. рис. 22.27) на выходе блока появится величина 4,2.

Вторым новым блоком, вводимым для обработки структур, является блок добавления, который служит для введения новых элементов в существующую структуру. Блок добавления имеет три входа, два из них такие же, как и у блока выбора, а третий используется для передачи в блок значения вводимого в

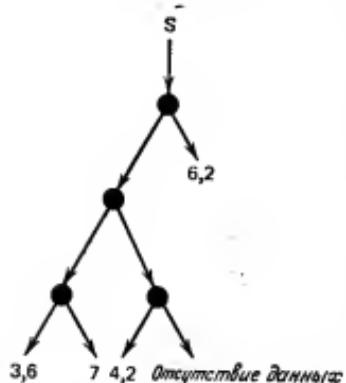


Рис. 22.27. Пример представления структуры.

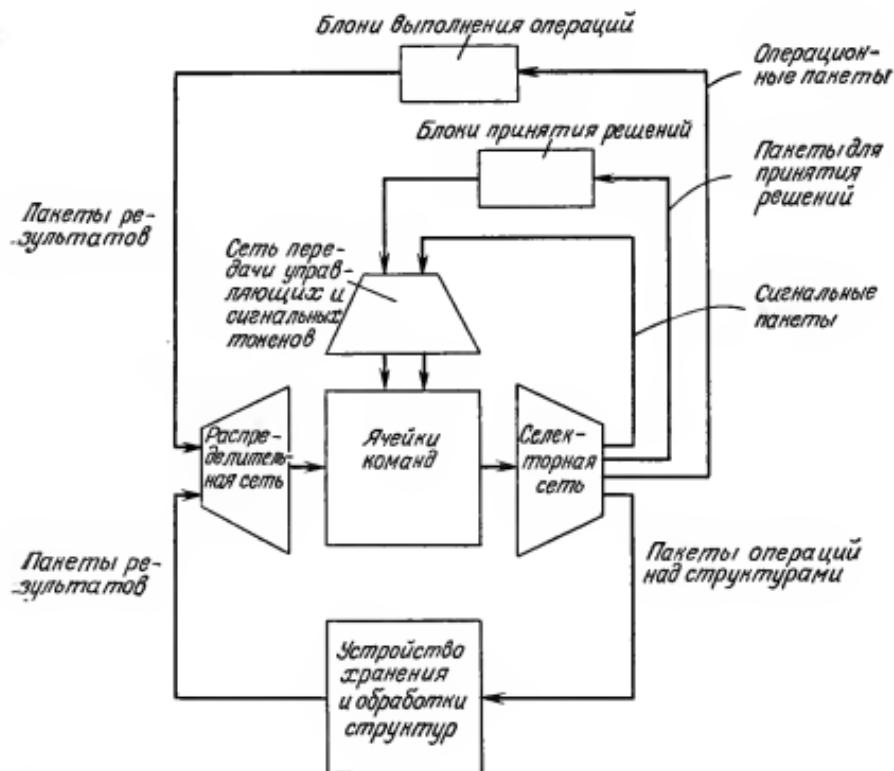


Рис. 22.28. Машина потоков данных, обрабатывающая структуры.

структуру элемента или адреса подсоединяемой структуры. На основании данных, поступающих на первые два входа, блок добавления находит нужный узел в дереве структуры. Содержимое этого узла заменяется данными, поступающими на третий вход. Так, при поступлении на входы блока добавления данных S, 010, 4,3 (где S соответствует рис. 22.27) значение 4,2 будет заменено на 4,3. Если на третий вход блока добавления подать адрес структуры, то последняя станет узлом исходной структуры.

Третьим новым блоком, вводимым для обработки структур, является блок удаления, имеющий два входа, на один из которых подается имя структуры, а на другой — двоичная строка. Адресуемый узел «стирается», т. е. ему присваивается значение «пусто». Предусмотрен также блок построения, который на основании двух входных данных (элементарных данных или структур) строит новую структуру, содержащую в качестве элементов входные данные. В результате работы блока на его выходной линии появляется адрес новой структуры.

На рис. 22.28 представлена обобщенная схема машины потоков данных, в которую введено *устройство хранения и обработки структур*. Когда селекторная сеть обнаруживает наличие операционного пакета, предназначенного для одного из блоков обработки структур, она передает этот пакет в устройство хранения и обработки структур. На выходе этого устройства появятся (в зависимости от типа блока) один или несколько пакетов результата, которые будут направлены в распределительную сеть.

На рис. 22.29 представлена схема устройства хранения и обработки структур. Для пояснения принципа его функционирования рассмотрим несколько примеров. Когда функционирует блок выбора, селекторная сеть направляет операционный пакет в распределительную сеть. Адрес структуры в команде выбора дает возможность извлечь слово из памяти хранения структур. Если значение слова определяет структуру, а не элементарные данные, то с помощью первого бита в двоичной строке, являющейся операндом для блока выбора, выбирается один из двух адресов подструктур, относящихся к данному узлу структуры. Этот адрес вместе с остатком двоичной строки помещается в новый пакет выбора и через селекторную сеть направляется вновь в распределительную сеть. Таким об-

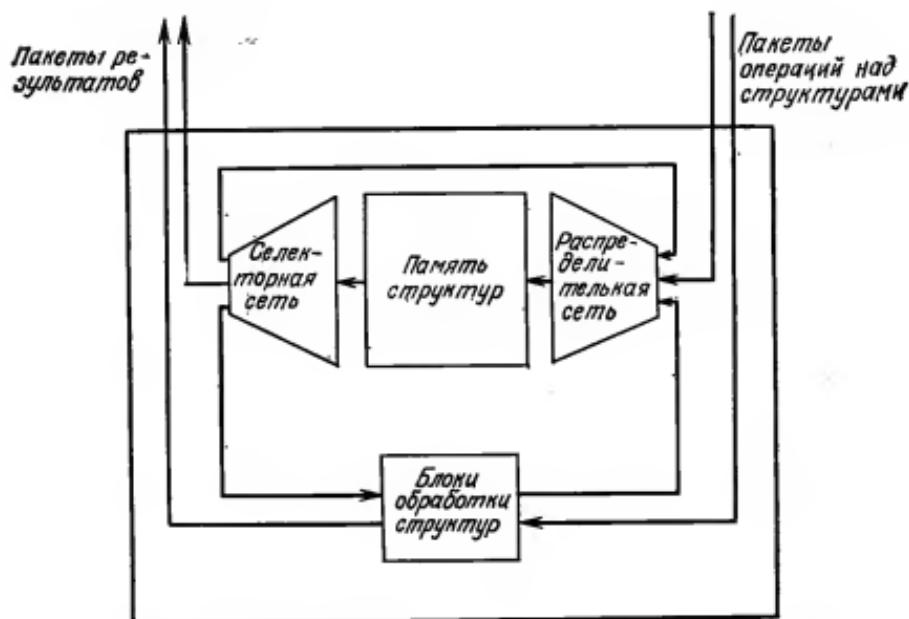


Рис. 22.29. Устройство хранения и обработки структур.

разом, в результате выполнения операции выбора может быть сформирована последовательность новых запросов на операцию выбора. Эти запросы последовательно обрабатываются памятью структур до тех пор, пока не встретится узел, содержащий элементарные данные. После этого формируется и выдается пакет, содержащий найденные элементарные данные и адрес пункта назначения, указанный в исходной команде выбора.

Другие операции над структурами выполняются одним или несколькими блоками обработки структур данных. Эти блоки оперируют содержимым памяти структур путем формирования запросов на обращение к памяти с целью извлечения ее содержимого или записи данных на хранение. Эти запросы имеют тот же формат, что и пакет запроса на операцию выбора, и так же направляются через распределительную сеть памяти структур. Результат, извлеченный из памяти структур, опять направляется в блок обработки структур. В конечном счете формируется пакет результата, который передается в общую распределительную сеть машины потоков данных.

## УПРАЖНЕНИЯ

22.1. Модифицируйте программу потоков данных, представленную на рис. 22.6, таким образом, чтобы после получения результата осуществлялся возврат в исходное состояние. (Для этого необходимо ликвидировать «застывшие» токены данных в соответствии с описанием рис. 22.11.)

22.2. Проследите ход выполнения программы, показанной на рис. 22.19, до ее завершения. Ответьте на следующие вопросы: произойдет ли останов программы после получения результата? Восстановит ли программа свое начальное состояние? Если начальное состояние не восстановится, предложите меры по корректировке программы, которые обеспечат выполнение этой процедуры.

22.3. Модифицируйте программу, представленную на рис. 22.15, используя для предотвращения тупиковых ситуаций сигналы обратной связи.

22.4. Каковы результаты выполнения команд `SELECT S,00,X` и `APPEND S,101,X`, где X представляет собой линию между командой `SELECT` (ВЫБОР) и командой `APPEND` (ДОБАВЛЕНИЕ), а структура S соответствует изображенной на рис. 22.27.

## ЛИТЕРАТУРА

1. Treleaven P. C., Exploiting Program Concurrency in Computing Systems, *Computer*, 12(1), 42—49 (1979).
2. Backus J., Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs, *Communications of the ACM*, 21(8), 613—641 (1978).
3. Peterson J. L., Petri Nets, *Computing Surveys*, 9(3), 223—252 (1977).
4. Dennis J. B., Misunas D. P., Leung C. K., A Highly Parallel Processor using a Data Flow Machine Language, Memo 134, Laboratory for Computer Science, MIT, Cambridge, MA, 1977.
5. Misunas D. P., A Computer Architecture for Data-Flow Computation, Laboratory for Computer Science, MIT, Cambridge, MA, 1978.

6. Dennis J. B., Misunas D. P., A Preliminary Architecture for a Basic Data-Flow Processor, Proceedings of the Second Annual Conference on Computer Architecture, New York, ACM, 1975, pp. 126—132.
7. Brock J. D., Operational Semantics of a Data Flow Language, MIT/LCS/TM-120, Laboratory for Computer Science, MIT, Cambridge, MA, 1978.
8. Gostelow K. P., Thomas R. E., A View of Data Flow, Proceedings of the 1979 NCC, Montvale, NJ, AFIPS, 1979, pp. 629—636.
9. Misunas D. P., Transcript—Workshop on Data Flow Computer and Program Organization, *Computer Architecture News*, 6(4), 6—22 (1977).
10. Ackerman W. B., Data Flow Languages, Proceedings of the NCC, Montvale, NJ, AFIPS, 1979, pp. 1087—1095.
11. McGraw J. R., Data Flow Computing, Software Development, Proceedings of the International Conference on Distributed Computing Systems, New York, IEEE, 1979, pp. 242—251.
12. Misunas D. P., Performance Analysis of a Data-Flow Processor, Proceedings of the 1976 International Conference on Parallel Processing, New York, IEEE, 1976, pp. 100—105.
13. Dennis J. B., Weng K. K. S., Application of Data Flow Computation to the Weather Problem, in D. J. Kuck, D. H. Lawrie, A. H. Sameh, Eds., High Speed Computer and Algorithm Organization, New York, Academic, 1977, pp. 143—157.
14. Dennis J. B., Packet Communication Architecture, Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, New York, IEEE, 1975, pp. 224—229.
15. Misunas D. P., Error Detection and Recovery in a Data-Flow Computer, Proceedings of the 1976 International Conference on Parallel Processing, New York, IEEE, 1976, pp. 117—122.
16. Misunas D. P., Performance of an Elementary Data-Flow Computer, Laboratory for Computer Science, MIT, Cambridge, MA, 1975.
17. Dennis J. B., Misunas D. P., A Computer Architecture for Highly Parallel Signal Processing, Proceedings of the 1974 ACM Annual Conference, New York, ACM, 1974, pp. 402—409.
18. Dennis J. B., The Varieties of Data Flow Computers, Proceedings of the Signal Processing, Proceeding of the 1974 ACM Annual Conference, New York, IEEE, 1979, pp. 430—439.
19. Davis A. L., A Data Flow Evaluation System Based on the Concept of Recursive Locality, Proceedings of the 1979 NCC, Montvale, NJ, AFIPS, 1979, pp. 1079—1086.
20. Arvind, Gostelow K. P., A Computer Capable of Exchanging Processors for Time, Proceedings of the 1977 IFIP Congress, Amsterdam, North-Holland, 1977, pp. 849—853.
21. Watson I., Gurd J., A Prototype Data Flow Computer with Token Labeling, Proceedings of the 1979 NCC, Montvale, NJ, AFIPS, 1979, pp. 623—628.
22. Treleaven P. C., Principal Components of a Data Flow Computer, Proceedings of the Fourth EURMICRO Symposium on Microprocessing and Microprogramming, Amsterdam, North-Holland, 1978, pp. 366—374.
23. Rumbaugh J. E., A Parallel Asynchronous Computer Architecture for Data Flow Programs, MAC-TR-150, MIT, Cambridge, MA, 1975.
24. Rumbaugh J. E., A Data Flow Multiprocessor, Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, New York, IEEE, 1975, pp. 220—223.
25. Rumbaugh J. E., A Data Flow Multiprocessor, *IEEE Transactions on Computers*, C-26(2), 138—146 (1977).
26. Davis A. L., The Architecture and System Method of DDM1: A Recursively Structured Data Driven Machine, Proceedings of the Fifth Annual Symposium on Computer Architecture, New York, ACM, 1978, pp. 210—215.
27. Plas A. et al., LAU System Architecture, A Parallel Data-Driven Processor

Based on Single Assignment, Proceedings of the 1976 International Conference on Parallel Processing, New York, IEEE, 1976, pp. 293—302.

28. Dennis J. B., Boughton G. A., Leung C. K., Building Blocks for Data Flow Prototypes, Proceedings of the Seventh Annual Symposium on Computer Architecture, New York, ACM, 1980, pp. 1—8.

29. Misunas D. P., Deadlock Avoidance in a Data-Flow Architecture, Laboratory for Computer Science, MIT, Cambridge, MA, 1975.

30. Misunas D. P. Structure Processing in a Data-Flow Computer, Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, New York, IEEE, pp. 220—223.

31. Ackerman W. B., A Structure Memory for Data Flow Computers, MIT/LCS/TR-186, Laboratory for Computer Science, MIT, Cambridge, MA, 1977.

32. Dennis J. B., Data Flow Supercomputers, *Computer*, 13(11), 48—56 (1980).

---

## ЧАСТЬ VIII

# ВОПРОСЫ, СВЯЗАННЫЕ С АРХИТЕКТУРОЙ СИСТЕМ

### ГЛАВА 23

#### ОПТИМИЗАЦИЯ И НАСТРОЙКА АРХИТЕКТУРЫ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

По завершении разработки первоначального проекта архитектуры вычислительной системы необходима оптимизация ее нижнего уровня, осуществляемая методами теории информации. Конечно, сам выбор варианта построения архитектуры можно рассматривать как процесс оптимизации. Здесь же речь идет об оптимизации нижнего уровня, т. е. о такой настройке архитектуры, при которой удается минимизировать количество битов, подлежащих пересылке между процессором и памятью при выполнении заданий программы. Благодаря этому удается сократить объем необходимой памяти, уменьшить время выполнения программ за счет более эффективного использования тракта память-процессор и — как полезный побочный эффект — отказаться от в определенном смысле произвольных ограничений на предельные значения кодов операций и адресов.

Оптимизация нижнего уровня сводится к отысканию наиболее эффективного метода кодирования информации. Существуют три основных подхода к решению этой проблемы: усовершенствование существующего набора команд путем введения в него новых команд, оптимизация представления отдельных компонентов команд (кодов операций, адресов, ссылок на операнды, данных) и оптимизация всей формы представления команды. Для успешного выполнения указанной оптимизации необходима детальная информация о характеристиках программ, включающая относительную частоту выполнения отдельных машинных команд и их сочетаний (например, зависимость выполнения одних команд от других или вероятность того, что после команды LOAD будет выполняться команда ADD) и относительную частоту присвоения переменным и адресам определенных значений. На первый взгляд может показаться, что единственным путем получения этих сведений является создание (физическая реализация) или моделирование машины, составление программ для этой машины с их последующим компилированием и изме-

рением искомых параметров этих программ. Однако далее будет показано, что необходимую информацию можно обеспечить без разработки машины и компиляторов, служащих для генерирования ее машинных программ.

Проблема сбора указанной информации осложняется тем, что возможны два подхода к измерению упомянутых выше частот: *статический* и *динамический*. Например, сведения о частоте использования команд определенного типа в программе могут быть получены путем подсчета количества команд, размещенных в памяти или генерируемых компилятором. В результате будет определена статическая частотная характеристика программы. Если эти данные использовать как исходные для процесса оптимизации проектируемой архитектуры, то этот процесс окажется ориентированным на сокращение требуемого объема памяти, но при этом не обязательно уменьшение времени выполнения программ. Следует, однако, учитывать тот факт, что сокращение рабочего пространства памяти, занятого программой, во многих ЭВМ приводит косвенно и к уменьшению времени выполнения. Кроме того, можно получить данные о частоте появления различных команд путем их подсчета во время выполнения. Такие частотные зависимости называются динамическими. Используя их в качестве исходных данных процесса оптимизации, можно определить выигрыш во времени выполнения программ, но не в объеме занимаемой ими памяти.

Казалось бы, что в каждом случае необходимо делать выбор только одного из параметров — времени выполнения программы или объема занимаемой памяти — в качестве критерия оптимизации. В действительности такая дилемма возникает редко. Многочисленные статистические данные указывают на тесную взаимосвязь статических и динамических частотных характеристик программ, что позволяет выбрать в качестве основы оп-

Таблица 23.1. Статическая и динамическая характеристики наиболее часто используемых команд Системы 360

Команда	Статическая частота появления команды, %	Команда	Динамическая частота появления команды, %
L	28,6	L	27,3
ST	15,0	BC	13,7
BC	10,0	ST	9,8
LA	7,0	C	6,2
SR	5,8	LA	6,1
BAL	5,3	SR	4,5
SLL	3,6	IC	4,1
IC	3,2	A	3,7

тимизации в каждом отдельном случае ту или другую характеристику и получить выигрыш сразу по двум критериям: времени выполнения программ и занимаемой ими памяти. В качестве наглядного подтверждения сказанному в табл. 23.1 по результатам анализа 19 программ приведены статические и динамические характеристики, определяющие частоту появления восьми наиболее часто используемых машинных команд Системы 360 [1].

## ОПТИМИЗАЦИЯ СИСТЕМЫ КОМАНД

Одно из возможных решений подобной оптимизации сводится к анализу использования в программах существующих машинных команд с целью создания новых команд, обеспечивающих сокращение времени выполнения программ и объема занимаемой ими памяти. В основу такого решения положено измерение частоты появления одного и того же сочетания следующих одна за другой двух или трех команд и создание одной команды, выполняющей те же функции и, следовательно, способной заменить эту группу команд.

В качестве примера рассмотрим следующую пару команд

SR регистр,регистр  
IC регистр,адрес

которая, как оказалось, встречается довольно часто в программах Системы 360 (статическая частота ее использования равна 2,7%, динамическая — 3,8% среди всех пар команд). Команда SR (ВЫЧИТАНИЕ регистровое) производит очистку регистра, а по команде IC (ЗАГРУЗКА СИМВОЛА в регистр) в регистр помещается величина, соответствующая представлению символа в памяти. Для улучшения состава набора команд можно было бы создать новую команду ОЧИСТКА РЕГИСТРА И ЗАГРУЗКА СИМВОЛА, занимающую меньший объем памяти и выполняемую быстрее, чем пара команд SR-IC. Можно ожидать, что такое тривиальное усовершенствование набора команд уменьшит размер программы в среднем на 1%, а время ее выполнения — на 1,5%.

Рассмотрим еще один пример. Анализ программ для Системы 370 показал, что компилятор часто генерирует следующую последовательность команд:

```
ST  REG,SAVE
L   REG,VARIABLE
LA  REG,N(REG)
ST  REG,VARIABLE
L   REG,SAVE
```

Эти команды предназначены для увеличения значения переменной целого типа VARIABLE на величину N. Поскольку в данной машине операции двоичной арифметики выполняются только над содержимым регистров, а число регистров невелико, требуется включение двух пар команд, обеспечивающих загрузку регистра и запись его содержимого в память. В модели 145 Системы 370 суммарное время выполнения этих команд равно 8,8 мкс, и они занимают 20 байт памяти. Если ввести новую команду INCREMENT, имеющую формат

INC VARIABLE,INCREMENT-VALUE

где INCREMENT-VALUE — приращение переменной VARIABLE, то она могла бы заменить все пять команд. Для размещения этой команды в памяти потребовалось бы 4 байта, ориентировочное время ее выполнения составило бы 2,7 мкс. Внимательный читатель может возразить, что посредством команды LA числовую величину можно увеличить на 4095, а с помощью команды INC — только на 16. Однако анализ частоты выполнения в программах различных подобных приращений показывает, что команда INC оказалась бы применимой в подавляющем большинстве случаев.

В машинах, построенных с использованием других архитектурных принципов, могут быть обнаружены аналогичные ситуации. Например, в ЭВМ со стековой организацией часто приходится выполнять последовательность команд

```
PUSH VARIABLE
PUSH 1
ADD
: STORE VARIABLE
```

Добавление не использующей стек команды

INCREMENT VARIABLE

дало бы очевидный выигрыш и во времени выполнения, и в объеме памяти.

#### ОПТИМИЗАЦИЯ КОМАНД УЧЕБНОЙ ПЛ-МАШИНЫ

Первоначальным и основным назначением учебной ПЛ-машины было использование ее как «инструмента» освоения техники оптимизации архитектуры вычислительной системы [3]. В гл. 5—7 описан «неоптимизированный» вариант этой машины; в данной главе рассматриваются некоторые пути усовершенствования ее архитектуры.

В качестве исходных данных для оптимизации использовались статистические характеристики 959 программ, прошедших

компилирование для выполнения на «неоптимизированном» варианте машины. Общее количество проанализированных операторов учебного языка ПЛ составило 37 443. Результат каждого изменения, вносимого в архитектуру оптимизируемой машины, количественно оценивался следующими показателями:

- A1 — число битов в командах программы;
- A2 — число битов в данных программы;
- A3 — число адресных ссылок к памяти при обращении к командам;
- A4 — число адресных ссылок к памяти при обращении к данным;
- A5 — число битов команд, извлекаемых из памяти в процессе выполнения программы;
- A6 — число битов в данных, извлекаемых из памяти в процессе выполнения программы.

Первым шагом анализа было определение частотных характеристик команд. Динамические и статические характеристики наиболее часто используемых команд приведены в табл. 23.2. (Команда NAME представляет команды SNAME и LNAME.)

Первым усовершенствованием, внесенным в рассматриваемую архитектуру, было изъятие команды LINE. Поскольку соответствие между машинной командой и номером оператора языка программирования носит статический характер, нет необходимости в постоянном аппаратном отслеживании этого соответствия в процессе выполнения программы. Вместо такого отслеживания можно воспользоваться информацией, содержащейся в формируемой программными средствами таблице соответствия выполняемых машинных команд и номеров операторов исходной программы, находящейся во внешней памяти. Эта простая модификация исходной архитектуры уменьшила в среднем показатель A1 на 14%, A3 на 8,7% и A5 на 12,2%.

**Таблица 23.2. Статическая и динамическая характеристики наиболее часто используемых команд учебной ПЛ-машины**

Команда	Статическая частота появления команды, %	Динамическая частота появления команды, %
NAME	30,3	28,3
EVAL	20,7	23,7
LINE	10,0	8,7
SWAP	4,8	4,1
POP	4,8	2,7

**Таблица 23.3. Наиболее часто используемые пары команд учебной ПЛ-машины**

Пары команд	Статическая частота появления пары команд, %
NAME, EVAL	19,8
NAME, NAME	8,9
EVAL, NAME	6,5
PARAM, SWAP	4,1
POP, NAME	3,6
STORE, POP	3,3

Следующим шагом анализа возможностей усовершенствования архитектуры являлось изучение частоты совместного использования пар команд. Статическая частота совместного использования шести наиболее употребительных пар показана в табл. 23.3.

Часто применяемые совместно команды NAME и EVAL служат для загрузки той или иной величины на вершину стека. Очевидное улучшение архитектуры машины достигается включением в существующий набор команд команды SLOAD (ЗАГРУЗКА короткая), выполняющей функции указанной пары, для замены пары команд SNAME и EVAL и команды LLOAD (ЗАГРУЗКА длинная) для замены пары команд LNAME и EVAL. Тогда вместо пары команд

```
SNAME 1,3
EVAL
```

предназначенных для загрузки переменной с адресом 1,3 на вершину стека, компилятор генерирует команду

```
SLOAD 1,3
```

Добавление команд SLOAD и LLOAD уменьшает показатель A1 в среднем на 23%, а также снижает значение показателей A3 и A5. Заметим, что хотя команды LOAD добавляются к набору команд машины, команды SNAME, LNAME и EVAL из него не изымаются, поскольку существуют ситуации, в которых необходимо использование каждой из них порознь.

Затем были произведены еще четыре подобные модификации архитектуры учебной ПЛ-машины. Было обнаружено, что в 72% случаев использования команды STORE за ней следует команда POP. Поэтому была добавлена новая команда STORED, выполняющая функции пары команд STORE и POP. Точно так же, согласно статистическим данным, в 71% случаев за командой SUBS следовала команда EVAL, что привело к введению команды SUBSEVEL. Дальнейший анализ пар команд показал, что за каждой командой DOTEST следует команда CRET, а за каждой командой DOINCR — команда CYCLE. В соответствии с этим выводом команды DOTEST и DOINCR были модифицированы таким образом, чтобы включить в себя функции команд CRET и CYCLE.

Затем усилия по улучшению архитектуры учебной ПЛ-машины были направлены на анализ возможностей средств адресации. Выяснилось, что среди команд типа NAME (SNAME и LNAME) 72,7% составляют команды SNAME. Целью оптимизации было увеличение процента использования команд SNAME, а также уменьшение длины команд SNAME и LNAME.

Прежде всего был проведен частотный анализ распределения адресов по лексическим уровням. Согласно результатам исследования [1], во-первых, более 80% всех адресов в командах ссылаются на внешний лексический уровень (лексический уровень 1, как и уровень 0, использовался в системных целях) и, во-вторых, более 90% адресов в командах конкретного лексического уровня являются обращениями к тому же лексическому уровню, т. е. большинство ссылок указывает на локальные переменные текущего уровня.

Указанные выше причины, а также неполное использование 8-битового поля кода операции (количество типов команд менее 256), явились основанием для внесения четырех изменений в учебную ПЛ-машину. Длина команды SNAME была уменьшена до 8 бит.

Вместо того чтобы, как это принято для всех команд, использовать для кода операции все 8 бит, полагают, что равенство нулю первой пары этих битов является признаком того, что данная команда — команда SNAME. Остальные 6 бит интерпретируются следующим образом. Если команда SNAME принадлежит лексическому уровню 1, то адресуемым лексическим уровнем считается уровень 1, а 6 бит — порядковым номером (0—63) адресуемого операнда этого уровня. Если команда не принадлежит уровню 1, первый из 6 бит указывает номер этого уровня: нулевое значение бита — текущий лексический уровень, единичное значение — лексический уровень 1. Остальные 5 бит содержат порядковый номер операнда данного уровня. (Если этот вопрос представляется читателю недостаточно ясным, следует обратиться к упр. 23.1 в конце данной главы.)

Второе изменение архитектуры учебной ПЛ-машинны заключалось в уменьшении длины новой команды SLOAD до 8 бит, выполненном по описанной выше схеме, только первые 2 бит команды SLOAD приняты равными 01. Результатом двух последних изменений, связанных с усовершенствованием средств адресации, явилось определение 16-битовых команд LNAME и LLOAD со следующей структурой формата: первые 5 бит — идентификатор типа команды, следующие 3 бит — указатель лексического уровня, последние 8 бит — порядковый номер операнда. Благодаря всем четырем изменениям показатель A1 (общее число битов команд программы) удалось уменьшить в среднем на 23%, а показатель A5 (число битов команд, извлекаемых из памяти) — на 13,5%. Кроме того, было установлено, что в среднем ~92% адресов в программе могут теперь быть представлены в более короткой форме — посредством команд SNAME и SLOAD.

Далее анализу было подвергнуто представление данных, и в частности констант, например константа 1 в операторе  $A = A + 1$ . Были установлены определенные закономерности. Во-первых, константы представляются в виде слов в стеке данных и адресуются на соответствующем лексическом уровне. Поскольку константам должны присваиваться порядковые номера, возникает необходимость использования длинных форм представления адресов лексического уровня. Во-вторых, каждый раз, когда требуется константа, ее необходимо извлекать из стека данных, а это увеличивает значение показателя A4. И, наконец, анализ частоты использования констант разных типов показал, что 72% всех констант имеют тип FIXED (двоичное целое число), причем значение половины констант этого типа равно 1, а 99% — менее 64. Следовательно, хранение константы как слова в стеке данных ведет к неоправданно большому перерасходу памяти, т. е. к хранению большого числа ведущих нулей, а это увеличивает значение показателя A2.

Результаты описанного анализа привели к заключению о целесообразности введения в набор команд новой 8-битовой команды LITERAL. Ее первые 2 бит, равные 10, указывают, что это — команда LITERAL. Следующие 6 бит представляют величину, которая интерпретируется как двоичное целое число, подлежащее загрузке на вершину стека. В результате лишь небольшую часть констант необходимо помещать в стек данных. Это позволяет уменьшить значения показателей A2, A4 и A6. Косвенно уменьшаются также и значения показателей A1 и A5. Это связано с тем, что уменьшение порядковых номеров операндов снижает потребность в «длинных» адресах: добавление команды LITERAL уменьшает диапазон порядковых номеров операндов на каждом лексическом уровне в среднем на 24%. Помимо этого, использование команды LITERAL привело к уменьшению размеров таблицы символов и сокращению времени начальной установки стека данных, выполняемой с помощью команды ENTER.

Суммарный эффект вышеописанных и некоторых других усовершенствований архитектуры учебной ПЛ-машины оказался поистине впечатляющим. В среднем число битов команд стандартной программы (показатель A1) сократилось на 51%, число обращений к памяти для извлечения команд (показатель A3) — на 37%, и число битов команд, извлекаемых из памяти (показатель A5) — на 58%. Кроме того, число обращений к памяти с целью извлечения данных (показатель A4) уменьшилось на 50%, а число битов данных, извлекаемых из памяти (показатель A6) — на 62%. Общий объем средней программы сократился на 23%, а число выполняемых команд — на 46%. Сравнение реальных программ, написанных для оптимизиро-

ванного варианта учебной ПЛ-машины, с аналогичными программами на языке ПЛ/1 для Системы 370 показало, что первые занимают в 13 раз меньший объем памяти и передают через интерфейс «память-процессор» в 3,5 раза меньше битов информации [3].

#### ПРИМЕРЫ ОПТИМИЗАЦИИ КОМАНД МАШИНЫ SWARD

В ходе разработки архитектуры SWARD был проведен анализ последовательностей команд, аналогичный рассмотренному выше. В качестве примера ниже приводится анализ взаимодействия команд проверки условий и передачи управления.

В первом варианте архитектуры проверка условий и передача управления выполнялись отдельными, независимыми друг от друга командами, как, например, в микропроцессоре iAPX 432. Так, вместо команды EQBF (проверка на равенство; переход, если ложно), в набор команд входили трехоперационные команды проверки отношений (команда EQ — проверка на равенство), помещающие логическое значение результата проверки отношения в операнд, и отдельно команда BF (переход, если «ложно»), проверяющая значение этого операнда на совпадение с заданной логической величиной. Однако анализ последовательностей команд показал, что за большинством команд проверки отношений следует команда BF, и поэтому операнд логического типа, используемый парой таких, следующих одна за другой команд, необходим только для установления временной связи между ними.

На основании этого было решено исключить указанный операнд логического типа, создав команды сравнения (например, описанные в гл. 15 команды EQBF, LTBF), которые объединяют операции проверки отношений и передачи управления. Более того, было предложено не просто добавить эти команды в существующий набор команд, а заменить ими команды проверки отношений и BF. Основаниями для такой замены являлись: во-первых, стремление свести к минимуму число команд с похожими функциями; во-вторых, предположение, что команды в прежнем виде больше не понадобятся; в-третьих, желание сократить количество команд для получения меньшего размера кода операции.

Было установлено, что все возможные варианты проверки отношений исчерпываются четырьмя различными конструкциями операторов. Эти операторы показаны в табл. 23.4 совместно с кодами команд, генерируемых машиной исходной и модифицированной архитектуры (т. е. после замены команд проверки и перехода командами сравнения).

**Таблица 23.4.** Последовательности команд сравнения и перехода в машине

Оператор или фрагмент оператора языка программирования	Последовательности команд исходного варианта	Последовательности команд модифицированного варианта
if B	BF B, \$X	EQBF B, 1, \$X
if X=Y	EQ B, X, Y	EQBF X, Y, \$X
if (X=Y and S=T)	BF B, \$X EQ B, X, Y EQ B1, S, T AND B, B1 BF B, \$X EQ B, X, Y	EQBF X, Y, \$X EQBF S, T, \$X
B:=X=Y;		MOVE B, 0 EQBF X, Y, \$X MOVE B, 1

Символ «=>» обозначает любую операцию отношения  
 X, Y, S, T — целые числа  
 B, B1 — логические величины  
 \$X — адрес перехода

В табл. 23.5 показаны те же конструкции с указанием их оценочной относительной частоты использования, полученной на основании анализа статистических данных по применению средств языка программирования. Данные, приведенные в таблице, наглядно демонстрируют получаемые при модификации преимущества в отношении как времени выполнения, так и объема используемой памяти. Согласно табл. 23.4, использование раздельных команд для проверки отношений и передачи управления дает выигрыш в случаях, когда предикат выражения IF является логической переменной или когда выражение в операторе присваивания является отношением. Однако эти ситуации, как видно из приведенных статистических данных, встречаются довольно редко и не могут быть основанием для сохранения раздельных команд проверки отношения и передачи управления.

Приведем другой пример усовершенствования набора команд машины SWARD. Ознакомимся с результатами анализа, аналогичного описанному выше, которому была подвергнута команда ITERATE. Первоначально у этой команды был еще один operand — приращение (шаг итерации). Анализ показал, что почти во всех случаях этот operand является литералом и его величина равна 1. Следующее по частоте использования значение этого литерала равно —1. (Не случайно в языке Ада в отличие от языков ПЛ/1 и ФОРТРАН шаг итерации может быть

Таблица 23.5. Сравнение двух вариантов обеспечения командами условной передачи управления

Оператор или фрагмент оператора языка программирования	Относительная частота использования в программе	Исходный вариант		Модифицированный вариант	
		Объем памяти, занимаемый кодом операции	Объем информации, передаваемой между памятью и процессором <sup>1)</sup>	Объем памяти, занимаемый кодом операции	Объем информации, передаваемой между памятью и процессором <sup>1)</sup>
if B	10	28	40	44	56
if X=Y	60	68	148	40	96
if (X=Y and S=	12 (если X=Y «истинно»)	140	320	80	192
=T)	12 (если X=Y «ложно»)	140	320	80	96
B:=X=Y;	3 (если значение результата «истинно»)	40	108	104	184
	3 (если значение результата «ложно»)	40	108	104	140
Среднее арифметическое	80	176	54	107	

<sup>1)</sup> Включая команды, теги, данные и результаты.

равен только 1 или —1<sup>1)</sup>.) В результате оптимизации команды ITERATE из расчета на наиболее часто используемое значение операнда последний был удален из команды, а шаг итерации принят равным 1. Кроме того, в набор команд машины была добавлена новая команда ITERATE-REVERSE с фиксированным значением шага итерации, равным —1. Прежний вариант команды ITERATE можно было бы оставить, но его исключили, руководствуясь тремя доводами, приведенными выше.

## ОПТИМИЗАЦИЯ КОДА ОПЕРАЦИИ

Хотя это может показаться и не очевидным, изложенные в предыдущем разделе методы совершенствования некоторых параметров вычислительной системы базируются на явным образом не сформулированном принципе устранения избыточности информации, хранимой в памяти машины. Для формального опре-

<sup>1)</sup> Это справедливо и для языка Паскаль. — Прим. перев.

делення и оценки таких понятий, как *избыточность* и *информационная емкость*, а также для разработки средств минимизации избыточности может быть привлечен аппарат теории информации [4].

Одним из достоинств этого аппарата является возможность теоретической оценки действительной информационной емкости сообщения. Эту оценку, обозначаемую буквой  $H$ , часто называют *количеством переданной информации источником*, *энтропией источника* или *мерой неопределенности сообщений источника информации*. В качестве отдельных частей сообщения можно рассматривать всю объектную программу, последовательность машинных команд, данные программы, адреса машинных команд и коды операций. Саму память, или точнее тракт процессор-память, можно считать каналом связи. Если части какого-то интересующего нас сообщения рассматривать как независимые друг от друга (выше было показано, что это неверно для последовательностей машинных команд или кодов операций, но предположим, мы хотим ограничиться только оптимизацией кодирования отдельных частей команды), то значение  $H$  может быть определено как

$$H = -\sum (P_i \log P_i),$$

где  $P_i$  — вероятность появления  $i$ -го символа сообщения. Поскольку информация представляется в битах, логарифмы берутся по основанию 2. Избыточность кодирования символов сообщения может быть определена следующим образом:

$$\text{Избыточность} = 1 - \frac{H}{\text{Действительный средний размер символа}}.$$

Диапазон возможных значений избыточности простирается от 0 (избыточность отсутствует) до 100% (бесконечная избыточность).

Если эта оценка применяется для оптимизации представления кода операций, то  $P_i$  обозначает вероятность появления  $i$ -го кода операции, а суммирование проводится по набору отдельных кодов операций. величина  $H$  — это среднее число битов информации в коде операции. Вероятность появления того или иного кода операции может быть либо статической (если объектом оптимизации является память, занимаемая кодами операций), либо динамической (если оптимизируется передача по тракту процессор-память). Однако, как уже отмечалось, существует сильная корреляция между статической и динамической частотами появления кодов операций, что позволяет произвольно выбирать в качестве предмета оптимизации одну из них и получать близкие к оптимальным результаты для другой.

Рассмотрим простой пример. Пусть машина имеет семь команд с именами A, B, C, D, E, F и G. Предположим, что для рассматриваемой машины длина кода операции фиксирована, тогда ее оптимальное значение равно 3 бит, а суммарная длина кодов операции программы из 1000 команд составит 3000 бит. Допустим, что вероятность появления семи подобных команд соответствует содержимому табл. 23.6. (Данные, приведенные в

Таблица 23.6. Вероятность появления команд

Команда	Вероятность появления, Р
A	0,50
B	0,30
C	0,10
D	0,03
E	0,03
F	0,02
G	0,02

таблице, не должны вызывать удивления: мы уже видели, что частоты появления различных команд могут существенно отличаться друг от друга.)

Величина Н для этой группы команд в предположении взаимной независимости появления команд, т. е. без учета частоты появления определенных последовательностей команд, равна 1,88. Это означает, что, хотя длина поля представления кода операции составляет 3 бит, число битов информации в каждом таком поле в

среднем равно 1,88. Избыточность в этом случае составляет 1—1,88/3, т. е. 37%. Очевидно, что должно существовать более эффективное представление кодов операций.

Повышение эффективности кодирования можно достичь, используя коды переменной длины, обратно пропорциональной частоте появления данной операции. Наиболее часто встречающиеся команды должны иметь самый короткий код операции, а редко выполняемые — самый длинный. Вместо того чтобы в каждом случае формулировать свой принцип формирования кодов операций, можно воспользоваться так называемым алгоритмом Хафмена, который обеспечивает оптимальное представление кода операции [4]. Действительно, одна из фундаментальных теорем теории информации доказывает, что при использовании кода Хафмена средний размер поля кода операции для рассмотренного случая принадлежит диапазону 1,88—2,21 бит. (Величина 2,21 является результатом вычисления выражения  $1,88 + 1/3$ , где 3 — размер поля представления кодов данной группы операций в том случае, когда длина поля фиксирована.) Кодирование, согласно алгоритму Хафмена, дает наилучшие практически возможные результаты, однако обычно не позволяет достичь желаемого предела, оцениваемого величиной Н, поскольку в этом случае коды операции должны быть представлены долями бита.

Кодирование, согласно алгоритму Хафмена, производится следующим образом. Строится двоичное дерево так, чтобы

подлежащая кодированию информация — коды операций — содержалась в концевых узлах графа. Первоначально в каждом узле указывается соответствующая ему частота использования кода операции данного типа. Эти узлы считаются «непокрытыми». Затем с целью получения одного непокрытого узла со значением частоты, равным 1,0, выполняется итеративная процедура «покрытия» узлов. Она состоит в следующем: отыскиваются два «непокрытых» узла с наименьшими значениями частот и «покрываются» путем их сведения в единый узел со значением частоты, равным сумме значений частот узлов, соединяемых звеньями с данным, только что сформированным и «непокрытым» узлом. Пример такого построения иллюстрирует рис. 23.1. Число звеньев от узла с частотой 1,0 до окончного узла представляет собой оптимальный размер кода операции, представляемого этим узлом. Помечая нулем и единицей соответственно левое и правое звенья, входящие в тот или иной узел, можно сформировать двоичное представление кода операции данного типа.

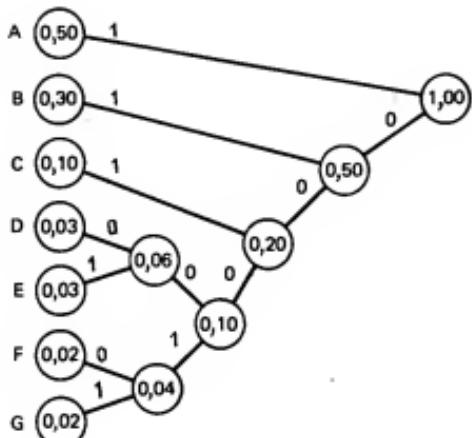


Рис. 23.1. Дерево кодирования операций по методу Хафмена.

Полученные данным методом коды операций для упомянутых выше команд A, B, C, D, E, F и G приведены в табл. 23.7. Заметим, что чаще других используемая команда A имеет 1-битовый код операции, а реже других выполняемые команды D — G — 5-битовый код. Средняя длина кода операции получается как сумма произведений частоты использования каждой команды на длину ее кода операции. В данном случае средняя длина оказывается равной 1,90 бит, избыточность составляет 1%, а число битов кодов операций в программе из 1000 команд равно теперь не 3000, а 1900.

Следует обратить внимание на однозначность дешифрирования кодов операций, т. е. на возможность для процессора безошибочного определения типа команды. Если бы код операции для B, например, был изменен с 01 на 10, то указанная однозначность оказалась бы утерянной, т. е. процессор был бы не способен различить команды A и B.

Несмотря на то что кодирование, согласно алгоритму Хаф-

Таблица 23.7. Кодирование операций по методу Хафмена

Команда	Вероятность появления, $P_j$	Код операции	Длина кода операции, бит
A	0,50	1	1
B	0,30	01	2
C	0,10	001	3
D	0,03	00000	5
E	0,03	00001	5
F	0,02	00010	5
G	0,02	00011	5

мена, обеспечивает оптимальное представление кодов операций, эта процедура никогда не использовалась. Основным препятствием на пути практической реализации такого решения является необходимость адресации к памяти машины с точностью до 1 бит. (Такой возможностью располагают вычислительные системы 432 фирмы Intel и 1700 фирмы Burroughs.) Другим затруднением на пути подобного решения является то, что принципиально каждый код операции может иметь свою длину, отличную от длины других кодов операций, что усложняет процесс декодирования этих кодов в процессоре. Однако в настоящее время с появлением программируемых логических схем матричного типа декодирование значительно упростилось.

Если кодирование по методу Хафмена оказывается трудоемким или в нем нет необходимости, можно принять компромиссное решение. Надо выбрать две или более фиксированные длины поля кода, и использовать самые короткие коды в наиболее часто выполняемых командах. Как будет показано на нескольких примерах, такое решение дает значительную экономию памяти по сравнению с использованием кодов операций постоянной длины.

Представим себе, что в некоторой гипотетической архитектуре коды операций могут иметь длину 2 или 4 бит. В табл. 23.8 показаны значения кодов операций. И в данном случае коды составлены таким образом, чтобы обеспечить однозначное декодирование команд. Средняя длина кода операции равна 2,20 бит, избыточность составляет 15%, а общее количество битов кодов операций в программе из 1000 команд оказывается равным 2200 вместо 3000.

Другим преимуществом такого метода кодирования, не отмеченным выше, является то, что он не накладывает ограничений на максимально допустимое число различных кодов операций. Указанные в табл. 23.8 илевые значения первых двух битов кода операции называют кодом переключения (escape со-

Таблица 23.8. Компромиссный вариант кодирования операций

Команда	Вероятность появления, $P_i$	Код операции	Длина кода операции, бит
A	0,50	11	2
B	0,30	10	2
C	0,10	01	2
D	0,03	0011	4
E	0,03	0010	4
F	0,02	0001	4
G	0,02	0000	4

де). Последний используется для указания того, что код операции имеет длину больше чем 2 бит. Используя этот же способ для индикации изменения длины кода операции применительно к следующей паре его битов, можно получить коды операций любой длины. Однако в таком случае комбинацию битов 00...00 нельзя использовать для обозначения кода операции. Следовательно, в табл. 23.8 команда G должна была бы иметь 6-битовый код операции 000011. Это увеличивает средний размер кода операции до 2,24 бит, что является незначительной платой за достигаемую при этом гибкость кодирования. Однако во многих случаях увеличения среднего размера кода операции не произойдет, поскольку число команд редко оказывается равным количеству всех возможных значений кода операции.

#### ОПТИМИЗАЦИЯ КОДА ОПЕРАЦИИ В ЭВМ В1700

В гл. 12 была описана оптимизация такого типа при рассмотрении архитектуры ЭВМ В1700, ориентированной на язык КОБОЛ. Коды операций имеют длину 3 или 9 бит. Семь наиболее часто используемых команд имеют 3-битовые коды операций, восьмое значение такого кода (из числа возможных для 3-битового кода) означает, что следующие 6 бит также являются частью кода операций. Теперь мы рассмотрим оптимизацию другой архитектуры ЭВМ В1700 — архитектуры, ориентированной на языке SDL [5].

Операционная система и компилятор ЭВМ В1700, написанные на языке SDL, выполняются этой машиной при условии, что ее архитектура ориентирована на этот язык. При разработке архитектуры проектировщики сопоставляли достоинства различных способов кодирования операции и выбрали для кодов три длины: 4, 6 и 10 бит. Из 16 возможных комбинаций значений первых четырех битов команды десять обозначают наибо-

Таблица 23.9. Сравнение разных вариантов кодирования операций в ЭВМ В1700 с архитектурой, ориентированной на язык SDL

Метод кодирования	Количество битов всех кодов операций команд операционной системы	Выигрыш, %	Затраты времени на декодирование команд, %
Метод с использованием длины поля 8 бит	301 248	0	0
Метод 4—6—10	184 966	39	2,6
Метод Хафмена	172 346	43	17,2

лее часто используемые команды, пять являются указанием на то, что код операции имеет длину 6 бит, и один указывает, что длина кода операции равна 10 бит.

Для оценки целесообразности такого выбора было проведено сравнение с кодированием операций 8-битовым кодом фиксированной длины и с кодированием согласно алгоритму Хафмена. В табл. 23.9 приведены сравнительные данные для этих трех способов кодирования в виде суммарного числа битов кодов операций, содержащихся в программах операционной системы, и относительных затрат времени на декодирование команд. Как следует из таблицы, выбранное решение оказалось вполне разумным: оно дает результаты, близкие к методу Хафмена по эффективности, и в то же время дополнительные затраты времени на декодирование команд незначительны. Набор кодов операций разной фиксированной длины (4, 6 и 10) уменьшает на 39% число битов, отводимых в операционной системе на коды операций. Это является существенным достоинством выбранного решения, поскольку в архитектуре, ориентированной на язык SDL, коды операции занимают почти треть общего объема программы.

#### ОПТИМИЗАЦИЯ КОДА ОПЕРАЦИИ В МАШИНЕ SWARD

В гл. 15 при определении набора команд машины SWARD указано, что код операции каждой команды формируется как совокупность того или иного количества 4-битовых групп. Доводы в пользу подобного кодирования операций с учетом частоты использования различных команд приводятся ниже.

Поскольку кодирование операций машины SWARD осуществлялось на той стадии проектирования, когда существовал только проект машины на бумаге, естественно возникновение вопроса, что использовалось в качестве источника информации о частоте употребления тех или иных команд. Эта информация

Таблица 23.10. Различные варианты кодирования операций в машине SWARD

Команда <sup>1)</sup>	Вероятность появления, $P_i$	Показатель ранжирования	Размер кода операции, бит			
			Метод Хаффмана	Метод А <sup>2)</sup>	Метод Б <sup>3)</sup>	Метод В <sup>4)</sup>
MOVE	24	1	2	4	4	4
ADD	14	2	3	4	4	4
BRANCH	11	3	3	4	4	4
EQBF	7	4	4	4	4	4
SUBTRACT	5	5	4	4	4	4
NEBF	4	6	4	4	4	4
LTBF	3	7	4	4	4	4
GTBF	3	8	8	4	4	4
CALL	2,5	9	5	8	4	4
MULTIPLY	2	10	6	8	4	4
ITERATE	2	11	6	8	4	4
LEBF	2	12	6	8	4	4
GEBF	2	13	6	8	4	4
ACTIVATE	1,5	14	6	8	4	4
DIVIDE	1,5	15	6	8	8	8
MOVESS	1,5	16	6	8	8	8
CONVERT	1,5	17	6	8	8	8
RETURN	1,5	18	6	8	8	8
COMPLEMENT	1,5	19	6	8	8	8
RANGECHECK	1,5	20	6	8	8	8
AND	0,5	21	7	8	8	8
OR	0,5	22	7	8	8	8
INDEX	0,5	23	7	8	8	8
ITERATEREV	0,4	24	8	8	8	8
SEND	0,4	25	8	8	8	8
RECEIVE	0,4	26	8	8	8	8
ABSOLUTE	0,3	27	8	8	8	8
LENGTH	0,3	28	8	8	8	8
CONCAT	0,2	29	10	8	8	8
Следующие 15 команд	1,5	30—44	10	8	12	16
Следующие 15 команд	1,5	45—59	10	8	16	16
Все остальные команды	0,5	60—64	10	8	20	16

<sup>1)</sup> Команда MARKER с 4-битовым кодом операции здесь не учитывается.

<sup>2)</sup> Метод А — для первых восьми команд размер кода операции равен 4 бит, для следующих 128 команд — 8 бит.

<sup>3)</sup> Метод Б — для первых 15 команд размер кода операции равен 4 бит, для следующих 15 команд — 8 бит и т. д.

<sup>4)</sup> Метод В — для первых 15 команд размер кода операции равен 4 бит, для следующих 256 команд — 16 бит.

получена «методом языковых фрагментов» [3], а именно путем анализа различных статистических данных [1, 3, 6, 10 и т. п.] о частоте использования операторов языка программирования. На основании этого анализа и сопоставления возможностей машины SWARD делалось заключение о том, какие машинные

команды порождаются теми или иными операторами языка программирования и какова частота генерирования таких команд.

Команды машины SWARD и полученные указанным образом оценки частоты их использования (вероятности появления) приведены в табл. 23.10. Значения частот несколько изменены для того, чтобы учесть команды, не имеющие непосредственных эквивалентов в конструкции того или иного оператора языка программирования высокого уровня. Неточность в оценке частот находится в пределах допустимых значений, поскольку для оптимизации имеет значение только относительная расстановка (ранжирование) команд по частоте их применения. Фактически, как будет показано ниже, небольшая неточность в ранжировании команд либо не влияет, либо оказывает лишь незначительное влияние на конечный результат.

Команда MARKER, хотя и имеет короткий 4-битовый код операции, исключена из рассмотрения, поскольку частоту ее использования определить затруднительно.

Прежде всего на основании имеющихся данных о частотах была рассчитана величина  $H$ . Ее значение оказалось равным 3,9 бит. Расчеты выполнены в предположении независимости использования команд, поскольку чем выше уровень языка программирования, тем меньше зависимость использования данной команды от предыдущих команд, образующих ту или иную последовательность команд в программе. Таким образом, для кода операции фиксированной длины (8 бит, или 2 токен) потребовалось бы поле представления, более чем в два раза превышающее теоретически необходимое.

Хотя была принята схема кодирования с использованием 4-битовых составных элементов кода операции (такую длину имеет минимальная адресуемая часть памяти), было проведено также кодирование методом Хафмена для выявления лучшего возможного представления кода операции. Размеры кодов операций, полученных по методу Хафмена, приведены в табл. 23.10. Средняя длина кода операции оказалась равной 4,1 бит. Были рассмотрены также три метода кодирования: метод А — 8/128, Б — 15/15/... и В — 15/15/256. Согласно методу А, длина кода операции восьми наиболее часто используемых команд равна 4 бит (1 токен), а следующих 128 команд — 8 бит (2 токена). Метод может быть реализован следующим образом. Если первый бит первого токена равен 1, код операции состоит из 1 токена (1XXX). Код операции всех остальных команд имеет вид OXXXXXXX.

В соответствии с методом Б длина кода операции 15 наиболее часто используемых команд равна 1 токен, следующих 15 команд — 2 токена и т. д. Если первый токен не заполнен

только нулями, то длина кода операции равна 1 токен. Если же все биты первого токена равны нулю, а у второго токена имеются отличные от нуля биты, то длина кода операции равна 2 токен, и т. д.

Согласно методу В, коды операций длиной 4 и 8 бит формируются аналогичным образом; длина всех других кодов операций полагается равной 16 бит.

За исключением того факта, что кодирование по методу Хафмена всегда дает оптимальные результаты, в настоящее время отсутствует какой-либо общепринятый способ сопоставления других методов кодирования с целью выбора наилучшего, помимо их экспериментальной проверки. В табл. 23.10 приведены данные о длине кодов операций, формируемой согласно трем перечисленным методам. Умножая длину каждого кода на соответствующую частоту использования команды и складывая полученные произведения, имеем среднюю длину кода операции: 5,2 бит для метода А, 4,85 бит для метода Б и 4,89 бит для метода В. Окончательный выбор был сделан в пользу последнего метода, по эффективности незначительно уступающего второму, но более простому по реализации.

## ОПТИМИЗАЦИЯ ПРЕДСТАВЛЕНИЯ АДРЕСОВ

Как уже упоминалось, процедура, аналогичная рассмотренной, может быть проведена и для других видов информации, таких, как данные или адреса данных и команд. Ниже рассматривается уменьшение избыточности информации, используемой для представления адреса.

Дополнительно к работам по оптимизации представления кода операции в ЭВМ B1700 фирмы Burroughs с архитектурой, ориентированной на язык SDL, была проведена оптимизация кодирования адресов данных [5]. Архитектура ЭВМ B1700 предусматривает адресацию лексического уровня.

Разработчики приняли решение обеспечить возможность использования до 16 лексических уровней и 1024 переменных на каждом уровне, что предполагает манипулирование на том или ином лексическом уровне 14-битовыми адресами — 4 бит для номера лексического уровня и 10 бит для порядкового номера или индекса переменной на этом уровне. После написания на языке SDL программ операционной системы и компиляторов был выполнен анализ частоты использования реальных адресов. Как можно было предположить, неравномерность оказалась значительной, что явилось следствием высокой степени избыточности представления адресов фиксированным числом битов.

Результаты описываемых исследований привели к изменению архитектуры машины — к отказу от применения адресов фиксированной длины и переходу к адресам, длина которых определяется частотой их использования. Адреса могут иметь длину 8, 11, 13 или 16 бит. Формат адреса состоит из трех полей: префикса, номера лексического уровня и порядкового номера адресата, принадлежащего этому уровню. Двухбитовое поле префикса определяет размер адреса. Длина поля номера лексического уровня может составлять 1—4 бит, длина поля порядкового номера — 5 или 10 бит. Анализ адресов данных в командах операционной системы показал, что если в первоначальной версии 9174 адреса занимали 128 436 бит, то в новой версии используется 94 900 бит, т. е. объем необходимой памяти сократился на 26%.

Оптимизация адресов команд осуществлялась аналогичным образом. Такой адрес состоит из номера сегмента программы и смещения команды внутри сегмента (в битах). Разработчики предусмотрели возможность использования до 1024 сегментов по 1 млн. бит в каждом, что приводит к применению 30-битовых адресов команд: 10 бит для имени сегмента и 20 бит для смещения команды внутри сегмента. Однако уже в первоначальном проекте машины была предусмотрена возможность использования 5- или 10-битовых имен сегментов и 12-, 16- или 20-битовых смещений. Как только выяснилось, что многие обращения к командам локализованы внутри сегмента, в архитектуру машины были внесены изменения, допускающие использование нулевого (длиной 0 бит) имени сегмента. Поскольку многие обращения производятся к началу других сегментов, было разрешено задание нулевого смещения. Таким образом, формат адреса команды состоит из трех полей, в которых располагаются: 3-битовый префикс, определяющий представление адреса; имя сегмента длиной 0, 5 или 10 бит и смещение длиной 0, 12, 16 или 20 бит (допускаются только восемь из всех возможных сочетаний форматов полей представления имен сегментов и смещений).

Чтобы оценить целесообразность использования этой интуитивно выбранной версии оптимизации, были собраны статистические данные о частоте использования различных адресов в программах, прошедших стадию компилирования. Оказалось, что предложенная версия достаточно близка к оптимальному решению и не требует изменений. Анализ 3767 адресов команд операционной системы показал, что они занимают 74 303 бит вместо 120 544 бит, которые потребовались бы для машины с байтовой структурой адресного пространства и фиксированной длиной адресов при том же диапазоне изменения адресов. Результирующая экономия памяти составила 38%.

## ОПТИМИЗАЦИЯ ПРЕДСТАВЛЕНИЯ АДРЕСОВ В МАШИНЕ SWARD

Архитектура машины SWARD может служить другим примером уменьшения избыточности в представлении адресов, однако способ, благодаря которому это достигнуто, отличен от способа, примененного разработчиками ЭВМ В1700. Оптимизация представления адресов возможна, если учесть следующие особенности данной архитектуры: во-первых, каждый модуль имеет свое собственное адресное пространство; во-вторых, адресное пространство данных отличается от адресного пространства команд; в-третьих, модуль сам определяет размеры адресов данных и команд, относящихся к нему.

Несмотря на то что первоначально только третья из указанных особенностей послужила основой для оптимизации представления адресов, на сокращение объема памяти, занимаемой адресами, влияют все указанные особенности. Рассмотрим машину, в которой все программы размещаются в едином обширном адресном пространстве или в нем по крайней мере одна любая программа помещается полностью, как в большинстве традиционных машин. Это означает, что посредством каждого адреса можно сослаться на любую ячейку памяти, однако, поскольку, например, семантика языков программирования накладывает ряд ограничений, существует сильная взаимосвязь между командами и адресами. Так, машинные команды, соответствующие подпрограмме на языке высокого уровня, ссылаются только на небольшую часть всех ячеек памяти, доступных для адресации. Следовательно, коды адресов содержат значительную избыточность. В машине SWARD значительная часть этой избыточности устранена. Поскольку каждый модуль располагает своим собственным адресным пространством, можно ограничиться меньшей длиной адреса данных.

Аналогичным образом, машине, размещающей команды и данные в одном адресном пространстве, присуща неоправданная избыточность в представлении адресов. Например, не все ячейки памяти с равной вероятностью оказываются адресуемыми командами передачи управления; вероятность указания в таких командах адресов ячеек памяти, содержащих данные, равна 0. Точно так же не все ячейки памяти равновероятно адресуемы командой ADD (СЛОЖЕНИЕ). В частности, в этой команде никогда не указываются адреса ячеек, содержащих команды. Таким образом, размещение команд и данных в различных адресных пространствах уменьшает избыточность и приводит к сокращению размера поля, необходимого для представления адресов.

Внимательный читатель может обнаружить еще два источника избыточности. Первый из них обусловлен тем обстоятель-

ством, что обращение к разным переменным в программе про-исходит с различной частотой. Следовательно, значения адресов не равновероятны и вычисление величины  $H$  показывает, что при фиксированной длине адресов имеет место избыточность. Второй и более существенный источник избыточности в представлении адресов обусловлен тем, что некоторые значения адресов никогда не используются, так как не указывают на начало полей представления данных ячеек памяти с данными. В качестве примера рассмотрим применительно к машине SWARD модуль с четырьмя ячейками. Длина первой ячейки составляет 4 токен; второй — 6 токен, третьей — 9 токен и последней — 4 токен. Тогда нумеруемые по порядку адреса ячеек могут иметь только следующие значения: 1, 5, 11 и 20. Однако принятая форма представления адресов позволяет задавать и другие значения последних: 2, 3, 4, 6 и т. д., что и объясняет появление значительной избыточности.

Такой тип избыточности анализировался в процессе разработки архитектуры SWARD. Были рассмотрены два способа адресации: прямая и косвенная. Избыточность при прямой адресации очевидна: поле адреса содержит смещение ячейки внутри адресного пространства модуля. При косвенной адресации в качестве адреса используется порядковый номер ячейки; соответствующий модуль содержит таблицу, называемую *таблицей смещения ячеек*, в которой для каждой ячейки указывается ее смещение внутри адресного пространства.

Поскольку ответ на вопрос, какой из способов предпочтительнее, не является очевидным, проводится сравнение указанных способов по следующим характеристикам: физическому размеру модуля; объему информации, передаваемому из памяти в процессор при выполнении модуля; количеству операций, выполняемых процессором для преобразования поля адреса в адрес физической ячейки; компактности размещения адресных ссылок (например, если процессор должен извлечь 2 токена данных, желательно, чтобы они размещались в смежных ячейках). При этом используются следующие обозначения:

$A$  — количество адресных полей в модуле (адресные поля размещаются в командах и ячейках «косвенный доступ к данным»);

$F$  — количество ссылок на операнды во время выполнения модуля;

$C$  — количество ячеек в модуле;

$S$  — размер (в токенах) адресного пространства модуля.

Функция  $CEIL$  используется для увеличения числа до ближайшего большего целого [ $CEIL(2) = 2$ ,  $CEIL(2.1) = 3$ ]. Все логарифмы берутся по основанию 16.

Что касается размера модуля, то для обоих способов адресации он одинаков, если не учитывать пространство памяти, занимаемое адресами и таблицей смещения ячеек. Постоянная для обоих способов величина в дальнейшем во внимание приниматься не будет. При косвенной адресации выражение

$$A \times \text{CEIL}(\log C) + C \times \text{CEIL}(\log S)$$

определяет размер пространства, занимаемого адресными полями и таблицей смещения ячеек. При прямой адресации подобная величина определяется согласно выражению

$$A \times \text{CEIL}(\log S).$$

Отношение этих двух выражений имеет следующий вид:

$$\frac{\text{CEIL}(\log C)}{\text{CEIL}(\log S)} + \frac{C}{A}.$$

Если значение полученного выражения меньше 1, то предпочтительнее косвенная адресация.

Поскольку только одно соотношение ( $S > C$ ) всегда истинно, ни один из методов не может быть выбран в качестве универсального для всех случаев. Но так как обычно  $S$  намного больше  $C$  (примерно на порядок), первое слагаемое приведенного выше выражения имеет, как правило, следующие значения: 1,0; 0,67 или 0,5. Если предположить, что для всех модулей величины  $C$  оказываются равномерно распределенными в диапазоне 2—300 и  $S = 10 \cdot C$ , то ожидаемым значением первого слагаемого будет 0,72. Очевидно выполнение соотношения  $A > C$ , так как в противном случае модуль будет содержать переменные, к которым не производится обращения. Если типовой модуль в среднем содержит не более трех обращений к каждой ячейке, то значение приведенного выше выражения окажется больше 1. Отсюда следует, что однозначный выбор способа адресации невозможен. Какой способ окажется более эффективным, имея в виду экономию памяти машины, зависит от характеристик конкретной программы.

При рассмотрении передачи данных из памяти в процессор оба способа оцениваются одинаковой для них составляющей, которая в дальнейшем при сопоставлении этих способов может быть исключена из рассмотрения (речь идет о пересылке кодов операций линтералов и содержащего ячеек). Поэтому сравнительный анализ сводится к оценке объема передаваемой адресной информации. При косвенной адресации эта величина определяется выражением

$$F \times \text{CEIL}(\log C) + F \times \text{CEIL}(\log S),$$

которое включает информацию о номере ячейки и ее смещении:

при каждой адресации к ячейке. При использовании прямой адресации объем информации равен

$$F \times \text{CEIL}(\log S).$$

Последнее выражение демонстрирует явное преимущество прямой адресации.

Что касается затрат на вычисление адреса, то при косвенном способе адресации требуется большее число операций процессора, поскольку номер ячейки должен быть уменьшен на размер элемента в таблице смещения ячеек для получения величины смещения. Таким образом, в данном случае использование прямой адресации также оказывается более желательным.

Сравнение затрат на передачу данных трудно выполнить с большой точностью, поскольку использование способа адресации, связанного с большим объемом передаваемых данных, может сопровождаться выполнением только небольшого числа операций извлечения данных из памяти, если они расположены в смежных ее областях; в то же время при использовании другого способа, возможно, меньшие по объему данные окажутся размещенными в разных, несмежных друг с другом участках памяти и для их извлечения потребуется больше операций. Следовательно, необходима оценка степени локальности адресных ссылок. Однако для способа косвенной адресации в этом нет необходимости, поскольку при использовании косвенной адресации требуется передача дополнительных данных, размещаемых отдельно от основной информации, так, элемент таблицы смещения ячеек расположен отдельно от поля адреса.

На основании проведенного анализа было отдано предпочтение способу прямой адресации (адресации посредством данных о смещении ячейки) перед способом косвенной адресации (адресации посредством данных о номере ячейки).

#### ОПТИМИЗАЦИЯ АРХИТЕКТУРЫ МИКРОПРОЦЕССОРА iAPX 432

Архитектура микропроцессора iAPX 432 является примером высокой степени оптимизации архитектуры вычислительной системы по многим параметрам.

Хорошо известно, что значительная часть данных, используемых в программах, является константами, причем по мере уменьшения значения константы вероятность ее использования резко возрастает; наиболее часто используемыми являются значения 0 и 1. Отказавшись от включения числовых литералов малой величины в команды вместо адресов операндов (как это сделано в машине SWARD), разработчики процессора iAPX 432 сосредоточили основное внимание на оптимизации манипулиро-

вания константами 0 и 1. Последние включены в набор команд, т. е. определяются специальными командами, использующими эти величины как неявные операнды. Это объясняет происхождение таких команд, как ZERO-ORDINAL, ONE-SHORT-ORDINAL, INCREMENT-INTEGER и EQUAL-ZERO-CHARACTER.

В поле класса длиной 4 или 6 бит указывается количество и длина операндов команды. Для наиболее употребимых команд (например, с двумя 16-битовыми или тремя 32-битовыми операндами) используется 4-битовое поле класса (см. табл. 18.1).

В поле формата указываются порядок и местоположение операндов — стек или какое-либо другое место. Если операндов нет, то это поле отсутствует; при наличии одного операнда оно содержит единственный бит и т. д.

Во многих случаях представление кода операции определяется частотой использования соответствующей команды. Так, команда DIVIDE-SHORT-ORDINAL, используемая намиого реже, чем команда DIVIDE-INTEGER, имеет 5-битовый код операции, а вторая из названных команд — 4-битовый код. Некоторые команды, например BRANCH, полностью определяются содержимым поля класса и поэтому не имеют поля кода операции.

Адреса также кодируются в соответствии с частотой их использования, хотя, возможно, более эффективно было бы организовать ссылки к локальным переменным контекста. Для выбора первых 16 сегментов в каждом из четырех входных списков доступа используется короткий 6-битовый селектор сегмента объекта. В редких случаях, когда этого оказывается недостаточно, можно использовать длинный 16-битовый селектор. Аналогично смещение может занимать 7 или 16 бит. Учитывая, что большинство переходов осуществляется к близлежащим «точкам» — командам, — адрес перехода может представлять собой 10-битовую относительную величину или 16-битовое абсолютное смещение в сегменте команд.

В литературе, список которой приводится ниже [7—13], содержатся примеры разных вариантов кодирования команд и адресов, используемых вычислительными системами различной архитектуры.

## УПРАЖНЕНИЯ

23.1. Поясните назначение следующих команд оптимизированной версии учебной ПЛ-машины:

- 10100100
- 00100011 на лексическом уровне 1
- 00000011 на лексическом уровне 1
- 00100011 на лексическом уровне 3
- 00000011 на лексическом уровне 3
- 01100011 на лексическом уровне 2

**23.2.** Пользуясь описанием рассмотренных выше команд SNAME, SLOAD и LITERAL учебной ПЛ-машины, попытайтесь определить коды операций других команд.

**23.3.** Выполните компилирование второй программы, написанной на языке учебной ПЛ-машины (гл. 6), в предположении, что в вашем распоряжении находится оптимизированный вариант машины. Сравните полученные результаты по занимаемому объему памяти с программой на рис. 6.4.

**23.4.** Положим, что набор команд некоторой машины ориентирован на язык программирования более высокого уровня, чем соответствующий набор команд другой машины. Поясните, большими или меньшими возможностями по оптимизации использования пар команд обладает первая машина.

**23.5.** Определите реальную среднюю информационную емкость кода операции, который используется в наборе из пяти команд, если вероятность появления команд в программе равна 0,5; 0,3; 0,1; 0,05 и 0,05 соответственно. Предполагается, что порядок следования команд не зависит от их типа. Какова будет избыточность при фиксированном 3-битовом коде операции? Как изменится действительная средняя информационная емкость при наличии зависимости между командами, т. е. если после появления какой-либо команды вероятность появления следующей команды не соответствует указанным выше значениям?

**23.6.** Хотя для машины SWARD сопоставлялись только три способа кодирования операций, существует много других решений этой задачи. Предложите представление кода операций для 14 команд посредством 1 токена, для 28 команд — посредством 2 токенов, а для 56 — посредством 3 токенов.

**23.7.** Сравните полученный в предыдущем упражнении способ кодирования с тремя способами, рассмотренными выше, и определите, какой из них лучше.

**23.8.** Укажите условия, при которых для машины с фиксированным 8-битовым кодом операции значение Н равно 8.

**23.9.** Определите, сколько 4-, 6- и 10-битовых кодов операций может быть задано при использовании метода кодирования операций 4—6—10 применительно к ЭВМ B1700 фирмы Burroughs, ориентированной на язык SDL.

**23.10.** Поясните способ кодирования ссылок команд перехода процессора iAPX 432 и укажите, как можно было бы усовершенствовать этот способ.

**23.11.** Поясните способ кодирования адресных ссылок на локальные переменные в процессоре iAPX 432 и укажите, как можно было бы усовершенствовать этот способ.

**23.12.** Как вы думаете, часто ли будет появляться приводимая ниже последовательность команд при работе процессора iAPX 432? Если часто, то укажите возможные пути оптимизации набора команд этой машины.

INCREMENT-SHORT-ORDINAL	I,1
EQUAL-SHORT-ORDINAL	I,UPPER,STACK
BRANCH-FALSE	STACK, \$TOP

## ЛИТЕРАТУРА

1. Alexander W. G., Wortman D. B., Static and Dynamic Characteristics of XPL Programs, *Computer*, 8(11), 41—46 (1975).
2. Svobodova L., Computer System Performance Measurement: Instruction Set Processor Level and Microcode Level, SEL-74-015, Digital System Laboratory, Stanford University, Stanford, CA, 1974.
3. Wortman D. B., A Study of Language Directed Computer Design, Ph. D. dissertation, Stanford University, Stanford, CA, 1972.
4. Ash R., Information Theory, New York, Wiley, 1965.
5. Wilner W. T., Burroughs B1700 Memory Utilization, Proceedings of the

1972 Fall Joint Computer Conference, Montvale, NJ, AFIPS 1972, pp. 579—586.

- 6. Elshoff J. L., An Analysis of Some Commercial PL/I Programs, *IEEE Transactions on Software Engineering*, SE-2(2), 113—120 (1976).
- 7. Hehner E. C. R., Matching Program and Data Representation to a Computing Environment, Ph. D. dissertation, University of Toronto, 1974.
- 8. Hehner E. C. R., Computer Design to Minimize Memory Requirements, *Computer*, 9(8), 65—70 (1976).
- 9. Hehner E. C. R., Information Content of Programs and Operation Encoding, *Journal of the ACM*, 24(2), 290—297 (1977).
- 10. Tanenbaum A. S., Implications of Structured Programming for Machine Architecture, *Communications of the ACM*, 21(3), 237—246 (1978).
- 11. Cook R. P., Lee I., An Extensible Stack-Oriented Architecture for a High-Level Language Machine, Proceedings of the International Workshop on High-Level Language Computer Architecture, University of Maryland, College Park, MD, 1980, pp. 231—235.
- 12. Stevenson J. W., Tanenbaum A. S., Efficient Encoding of Machine Instructions, *Computer Architecture News*, 7(8), 10—17 (1979).
- 13. Johnson S. C., A 32-bit Processor Design, Computing Science Technical Report 80, Bell Labs, Murray Hill, NJ, 1979.

---

## ГЛАВА 24

### ПРАКТИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОЕКТИРОВАНИЮ АРХИТЕКТУРЫ ЭВМ

До сих пор в этой книге основное внимание было сосредоточено на принципах организации архитектуры ЭВМ, позволяющих сократить традиционные семантические разрывы. Для иллюстрации этих принципов использовались описания реально существующих машин или проектов таких машин. Однако мало что было сказано о методологии самого процесса проектирования архитектуры ЭВМ. Поэтому представляется естественным завершить книгу кратким обсуждением некоторых задач и средств проектирования архитектуры вычислительных машин.

#### ВОПЛОЩЕНИЕ ПРИНЦИПА КОНЦЕПТУАЛЬНОГО ЕДИНСТВА В АРХИТЕКТУРЕ

В вычислительной технике термин «концептуальное единство» впервые занял прочное место в лексиконе разработчиков внешних интерфейсов средств программного обеспечения ЭВМ [1], хотя это понятие в равной мере применимо как к архитектуре ЭВМ, так и к другим отраслям знаний. Понятие «концептуальное единство» является синонимом понятий однородности, унифицированности, единобразия. Применительно к архитектуре ЭВМ это означает завершенность и регулярность структуры, сведение к минимуму особых, или исключительных, ситуаций. При знакомстве с архитектурой, в основу которой положен указанный принцип, должно создаваться впечатление, что она задумана, спроектирована и реализована как бы одним человеком.

Воплощение принципа концептуального единства в архитектуре ЭВМ имеет большое значение по крайней мере по четырем причинам. Во-первых, благодаря высокой степени однородности структуры и минимизации особых ситуаций в системе уменьшается количество возможных ошибок при ее реализации. Во-вторых, удешевляется стоимость составных частей системы, например процессора, поскольку высокая степень регулярности

выполняемых системой функций позволяет реализовать их, используя один и те же аппаратные средства (в режиме разделения времени). В-третьих, значительно упрощается программирование, или генерирование кодов. Действительно, отклонение от действия того или иного принципа организации на всю систему полностью сопряжено с появлением в системе большого числа особых ситуаций, что приводит к увеличению времени программирования, возрастанию вероятности появления ошибок и усложнению процесса отладки программ на языке ассемблера. Если концептуальное единство организации вычислительной системы не является ее неотъемлемой характеристикой, т. е. если принцип реализации локально (в той или иной части системы), это может привести к существенному увеличению части компилятора, выполняющей функции генератора объектного кода. В-четвертых, большое количество особых ситуаций, предусматриваемых архитектурой системы, может вызвать снижение эффективности программ из-за возникновения трудностей при реализации оптимизирующих возможностей компиляторов в процессе обработки особых ситуаций.

Такие принципы организации архитектуры ЭВМ, как теговая память и соответствующий набор команд, инвариантных к типу обрабатываемых данных, одноуровневая память, объекты и общий для всех команд механизм адресации, являются путями решения проблемы концептуального единства. Однако лучше всего понять важность принципа концептуального единства можно, анализируя примеры, где этот принцип не соблюден. Приведем несколько примеров указанного типа, принятых при разработке архитектуры Системы 370.

1. Машина располагает командами преобразования целых двоичных чисел в целые десятичные и обратного преобразования, но аналогичные команды преобразования данных другого типа (например, чисел с плавающей точкой, десятичных правильных дробей, символьных строк) отсутствуют.

2. Многие команды, использующие набор регистров общего назначения, должны оперировать двумя смежными регистрами. В команде требуется указание регистра с младшим номером, причем этот номер должен быть четным.

3. Если при выполнении команды CONVERT-TO-BINARY возникает переполнение, регистрируется особая ситуация «деление с плавающей точкой».

4. Существуют два представления двоичных целых чисел: с помощью слова (32 бит) и полуслова (16 бит). Почти все команды, оперирующие словами (например, LOAD, STORE, ADD, SUBTRACT, MULTIPLY), имеют соответствующие команды для работы с полусловами; исключение составляет команда DIVIDE.

5. Некоторые команды (например, команда MOVE CHARACTER LONG) используют четыре регистра общего назначения, что усложняет решение задачи оптимизации распределения регистров.

6. Название почти всех регистров общего назначения отражает их функциональную задачу. Однако имеется команда TRANSLATE-AND-TEST, которая может работать только с регистрами 1 и 2.

7. Существует команда HALVE (ДЕЛЕНИЕ ПОПОЛАМ) для чисел с плавающей точкой, но аналогичные команды для двоичных целых или десятичных чисел отсутствуют.

8. Вследствие выполнения одних команд устанавливается признак результата, после выполнения других не устанавливается; критерий разделения команд по этому признаку носит почти случайный характер. Так, при выполнении команд сложения и вычитания признаку результата присваивается значение, а при реализации команд умножения и деления он игнорируется; при выполнении команды LOAD-POSITIVE признак устанавливается, а при выполнении команды LOAD — нет. После сложения двоичных или десятичных целых признак результата может быть проверен на переполнение, а после сложения чисел с плавающей точкой такая проверка не производится.

#### ОРТОГОНАЛЬНОСТЬ ПРИНЦИПОВ АРХИТЕКТУРЫ

В то время как принцип концептуального единства гласит: «Объекты должны быть однородными», принцип ортогональности формулируется следующим образом: «Объекты должны иметь свои отличительные характеристики». Эти два принципа не противоречат друг другу, поскольку по-разному определяют понятие «объекты». Применительно к требованию ортогональности понятие «объекты» означает типы данных, команды и другие специфические элементы архитектуры ЭВМ. Что касается концептуального единства, то в данном случае под объектами подразумеваются принципы и средства, с помощью которых обеспечивают манипулирование типами данных, командами и т. п.

Свойство ортогональности предполагает «взаимную перпендикулярность» принципов организации архитектуры ЭВМ и преследует перечисляемые ниже цели:

- 1) поддержание числа базовых принципов архитектуры на некотором минимальном уровне (в разумных пределах);
- 2) достижение максимальной независимости этих принципов;
- 3) избежание «излишеств» в средствах, которые реализуют принципы, заложенные в основу архитектуры машины.

Архитектуре, принципы организации которой не удовлетворяют в должной степени требованию ортогональности, присущее «перекрытие» этих принципов. Следствием этого является наличие в машине не имеющих особой практической пользы, но красиво выглядящих функциональных взаимосвязей в виде, например, 17 различных способов очистки регистра или приращения его содержимого. Если же принципы архитектуры в значительной мере удовлетворяют требованию ортогональности, то соответствующая машина того же уровня сложности и стоимости способна предоставить большие функциональные возможности.

#### АДЕКАВТАНСТЬ АРХИТЕКТУРЫ ТРЕБОВАНИЯМ «ВНЕШНЕГО МИРА»

Содержание всех предыдущих глав книги должно было постепенно привести читателя к заключению о необходимости формирования архитектуры ЭВМ на основе требований внешней среды, которую машине надлежит «обслуживать». Так, архитектура должна определяться используемым языком программирования: его структурой, особенностями применения, типом решаемых задач. Большинство свойств машин традиционной архитектуры только в незначительной степени удовлетворяет этим требованиям.

Характерным примером является вопрос о признаках результата. Можно утверждать, что они не только малополезны в языках высокого уровня, но просто нежелательны. В программе на языке высокого уровня признаки результата используются только при вычислении логических выражений — в операторах IF и операторах присваивания. Однако признак результата устанавливает ряд команд, не связанных с обработкой логических выражений, например в Системе 370 это команды ADD, LOAD-POSITIVE и многие другие. Компилятор эти признаки результата не использует, но сам факт их установки увеличивает затраты, связанные с практической реализацией команд. Так, анализ программ для машины PDP-11 [2] показывает, что при использовании команд, основным назначением которых не является формирование признака результата (в эту категорию не входят команды сравнения), значения признака результата не применяются на протяжении 92% времени работы программы.

Другим примером неадекватности архитектуры традиционных машин требованиям обслуживаемого ими «внешнего мира» является описываемая в гл. 23 раздельная реализация операций сравнения и передачи управления.

Еще одним примером указанной выше неадекватности может служить ситуация, когда требуется вычислить логическое

выражение и использовать полученный результат для каких-то целей, не связанных с передачей управления. В подобных случаях механизм оперирования признаком результата оказывается весьма неудобным. Обычно признак результата является частью слова состояния процессора и не может обрабатываться как данные. Например, в Системе 370 для использования признака результата в качестве данных необходимо либо применение последовательности команд условного перехода к командам, формирующим признак результата как некоторую величину, соответствующую выбранной ветви перехода, либо генерирование прерывания, либо выполнение «фiktивной» команды BRANCH-AND-LINK, сохраняющей в регистре признак результата.

Команды сдвига — еще один пример возможностей, предоставляемых архитектурой ЭВМ, но находящих лишь незначительное практическое применение. При наличии команд умножения и деления команды сдвига вряд ли будут широко использоваться, если основным средством программирования являются языки высокого уровня. Кстати, общим заблуждением многих разработчиков и типичной ошибкой, которую можно обнаружить в компиляторе, является представление о том, что сдвиг вправо эквивалент делению [3]. В арифметике, построенной на двоичном дополнительном коде, сдвиг вправо дает результат, совершиенно не согласующийся с правилами целочисленной арифметики для большинства языков программирования. Так, например, при делении  $-1$  на  $2$  путем сдвига результат равен  $-1$ , в то время как, согласно требованиям большинства языков программирования, результат должен быть равен  $0$ .

#### ОПТИМИЗАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ СРЕДСТВ ПО КРИТЕРИЮ ЧАСТОТЫ ИХ ИСПОЛЬЗОВАНИЯ

В предыдущей главе было показано, что частота использования команд, данных и адресов распределена по диапазону возможных значений крайне неравномерно. В подобных случаях обычно целесообразен поиск оптимальных решений, даже если это приводит к определенным нарушениям требований единообразия и ортогональности архитектуры ЭВМ.

Для оптимизации необходимо располагать статистическими данными относительно того, как будет использоваться машины данной архитектуры (или как используются уже существующие машины той или иной архитектуры либо языки программирования).

При анализе машинных команд основное внимание должно уделяться следующим наиболее употребительным операциям:

1. Копирование значения константы или переменной в поле другой переменной (например,  $A := B$ ).

2. Добавление к значению переменной некоторой величины, являющейся чаще всего небольшой по значению константой (обычно добавляется 1).

3. Сравнение двух величин на равенство и условный переход к точке программы, расположенной на небольшом расстоянии за точкой ветвления. Часто один из операндов сравнения является константой 0 или 1.

Конечно, оптимизация не должна ограничиваться только этими операциями, однако их следует иметь в виду при выборе направления работы.

Обычно оптимизация влечет за собой появление дополнительных особых ситуаций, что на первый взгляд не согласуется с некоторыми предыдущими рекомендациями. Однако следует стремиться к стиранию принципиальных различий между так называемыми особыми и типичными ситуациями. Пример прямо противоположного решения — команда приращения в ЭВМ PDP-11 [2]. Хотя может показаться, что она представляет собой оптимизированный вариант команды сложения, результат ее выполнения не полностью совпадает с результатом обычного прибавления 1. Побочные эффекты, производимые этими двумя командами, в частности установка флага переноса, различные.

Отметим, что оптимизация, базирующаяся на неравномерности частоты использования вычислительных средств, как и многие другие идеи, может быть отнесена к предложениям, высказанным фон Нейманом: «Мы хотели бы ввести в машину в виде электронных схем только такие логические структуры, которые либо необходимы для функционирования полноценной системы, либо очень удобны, поскольку часто используются...»

#### ПРОЕКТИРОВАНИЕ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ С УЧЕТОМ ЕЕ ВОЗМОЖНОГО РАСШИРЕНИЯ

Возможность расширения архитектуры вычислительной системы при соблюдении требования совместимости создаваемых моделей является еще одной важной проблемой, общие рекомендации по решению которой сформулировать нелегко.

Помимо прочего способность системы к расширению предполагает отказ от наложения произвольных ограничений на количество типов команд и данных, а также на максимальные значения адресов. Некоторые приемы оптимизации времени выполнения программ и объема занимаемой ими памяти, рассмотренные в гл. 23 (например, оптимизация представления кодов операций, адресов и данных, основанная на неравномерности

распределения частот их использования), могут оказаться полезными и при решении проблемы расширения архитектуры вычислительной системы.

Конечно, при решении этого вопроса, как и многих других, необходимо «чувство меры». Стремление к предоставлению вычислительной системе наибольших возможностей по ее расширению должно согласовываться со всеми другими целями для достижения оптимального сбалансированного решения.

#### НЕЗАВИСИМОСТЬ АРХИТЕКТУРЫ СИСТЕМЫ ОТ УСЛОВИЙ ЕЕ КОНКРЕТНОЙ РЕАЛИЗАЦИИ

Идеальным следует считать такое решение, при котором архитектура системы накладывает лишь незначительные ограничения на процессор, реализующий ее основные функции. Например, архитектура должна допускать практическую реализацию широкого набора вычислительных машин различной производительности и стоимости — семейства совместимых процессоров. По-видимому, архитектура высокого уровня в большей мере отвечает этому требованию, чем архитектура традиционных систем. Например, команды, которыми располагает архитектура высокого уровня, обладают большими возможностями для распараллеливания вычислительного процесса. При этом допускается и последовательное выполнение этих команд на более дешевых и менее производительных моделях вычислительной системы при сохранении возможностей создания варианта системы с высоким уровнем параллелизма вычислений.

#### НЕЗАВИСИМОСТЬ АРХИТЕКТУРЫ СИСТЕМЫ ОТ ТЕХНОЛОГИЧЕСКОЙ БАЗЫ

Многие системы проектируются в расчете на длительное использование и поэтому должны быть приспособлены к условиям изменяющейся технологии и экономики. Так, объем памяти 16М байт представлялся вполне разумным верхним пределом во время разработки Системы 360, но с тех пор 24-битовая адресация стала «занозой в теле» фирмы IBM. Учитывая, что уже в те годы существовали идеи использования виртуальной памяти и режима разделения времени, что 32-битовая адресация не менее сложным образом реализуется архитектурой этих машин, а также то обстоятельство, что технология производства интегральных схем была уже близкой перспективой, нельзя считать разумным решение об использовании 24-битовой адресации для больших систем.

Поскольку технология изготовления логических схем и запоминающих устройств развивается довольно быстро, следует

избегать решений, жестко ориентированных на использование технологической базы, доступной к моменту практической реализации системы. Это хорошо согласуется с тем, что на этапе разработки архитектуры системы такие ее характеристики, как потенциальная адресация и одноуровневая память, не требуют указания объема, типа и характеристик физически реализуемой запоминающей среды.

#### ФОРМАЛИЗОВАННОЕ ОПИСАНИЕ АРХИТЕКТУРЫ

Еще одним важным вопросом проектирования архитектуры является использование формализованного языка описания спецификаций системы как дополнение к обычному словесному описанию технических характеристик ЭВМ. Это дает следующие преимущества: во-первых, формализованное описание системы более точное и определенное, чем словесное описание; во-вторых, если формализованное описание является семантическим или алгоритмическим определением архитектуры, его можно использовать как руководство при проектировании процессора на этапе реализации данной архитектуры; в-третьих, если язык описания пригоден для работы на ЭВМ (его выполняем), то он может служить моделью архитектуры для изучения ее рабочих характеристик и проверки ее практической реализуемости; в-четвертых, наличие подобного языка описания позволяет использовать технику корректировки программ для оценки алгоритмов архитектуры проектируемой машины.

Наиболее известным языком описания является язык ISP [5]. К сожалению, он в основном ориентирован на описание синтаксиса архитектуры, например форматов команд и данных; для описания ее семантики этот язык мало подходит. Можно назвать еще несколько подобных языков, применявшихся для решения аналогичных задач. Так, учебная ПЛ-машина была определена с помощью процедурного языка MDL [6], Система 360 — с использованием языка APL [7], а для архитектуры машины SWARD был разработан расширенный диалект языка ПЛ/1. В процессе создания алгоритмического описания машины SWARD было обнаружено 13 требующих решения проблем, которые остались незамеченными при словесном описании машины.

#### МЫСЛЕННОЕ КОМПИЛИРОВАНИЕ КАК СРЕДСТВО ПРОЕКТИРОВАНИЯ АРХИТЕКТУРЫ

Много раз в этой книге мысленное компилирование программы, написанной на языке высокого уровня, использовалось как средство изучения архитектуры машины. Этот прием может

быть очень эффективным при разработке архитектуры, поскольку дает возможность анализировать возможные варианты и выявлять просчеты.

#### **РЕАЛИЗУЕМОСТЬ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ МАШИНОЙ ОПРЕДЕЛЕННОЙ АРХИТЕКТУРЫ**

Для машины фон Неймана обычно справедливо следующее: если машина допускает компилирование программы, написанной на каком-либо языке программирования, то на этой машине можно компилировать программы, создаваемые на любых существующих языках программирования. Таким образом, при разработке традиционной архитектуры вопрос может стоять лишь об эффективности компилирования программ и очень редко или вообще никогда — о принципиальной возможности компилирования с конкретного языка.

Однако для машин с архитектурой высокого уровня вопрос о возможности компилирования с различных языков является новым и важным. Поскольку в этих машинах устранена опасная универсальность, присущая машинам фон Неймана, а современные языки разработаны с расчетом на работу с машинами фон Неймана, вполне может оказаться, что компилирование каких-либо языковых конструкций окажется невозможным.

Вопрос о правильности использования языка программирования решается выяснением, допускает ли данная архитектура компилирование программ, написанных на этом языке. Неформальным методом, который может быть использован для этой цели, является проверка каждой конструкции языка (например, каждого типа данных, оператора, знака операции) на соответствие одной или нескольким конструкциям архитектуры, т. е. на реализуемость ими. В частности, при сопоставлении возможностей языка ПЛ/1 и учебной ПЛ-машины становится ясно, что разработка для последней компилятора языка ПЛ/1 невозможна, поскольку некоторые важные элементы структуры языка ПЛ/1 не могут быть реализованы. При сопоставлении возможностей языка ФОРТРАН и машины SWARD видно, что на последней нельзя выполнить некоторые типы оператора EQUIVALENCE, например объявление эквивалентных переменных целого и вещественного типов.

#### **ОТКАЗ ОТ КОЛЛЕКТИВНОГО ПРИНЯТИЯ РЕШЕНИЙ**

Провал многих проектов разработки архитектуры ЭВМ, новых языков программирования и операционных систем связан с коллективным подходом к выработке решений. Хотя участие неко-

торого комитета в процессе работы над проектом представляется вполне разумным (например, с целью внесения новых идей, изучения первоначальных решений, оценки возможных вариантов), принятие окончательных решений, как и составление спецификаций законченного проекта, в идеальном случае должно осуществляться одним лицом. «При проектировании суперкомпьютеров ни один коллектив не добьется таких успехов, как один высококвалифицированный энергичный разработчик. Ясное понимание задач и цельность видения оказываются в ходе большой работы более весомыми, чем ошибки, которые может допустить один человек» [8].

## ЛИТЕРАТУРА

1. Brooks F. P., *The Mythical Man-Month, Essays on Software Engineering*, Reading, MA, Addison-Wesley, 1975.
2. Russell R. D., *The PDP-11, A Case Study of How Not to Design Condition Codes*, Proceedings of the Fifth Annual Symposium on Computer Architecture, New York, ACM, 1978, pp. 190—194.
3. Steele G. L., Jr., *Arithmetic Shifting Considered Harmful*, *SIGPLAN Notices*, 12(11), 61—69 (1977).
4. Burks A. W., Goldstine H. H., von Neumann J., *Preliminary Design of the Logical Design of an Electronic Computing Instrument*, in A. H. Taub, Ed., *Collected Works of John von Neumann*, Vol. 5, New York, Macmillan, 1963, pp. 34—79.
5. Bell C. G., Newell A., *Computer Structures, Readings and Examples*, New York, McGraw-Hill, 1971.
6. Wortman D. B., *A Study of Language Directed Computer Design*, Ph. D. dissertation, Stanford University, Stanford, CA, 1972.
7. Falkoff A. D., Iverson K. E., Sussenguth E. H., *A Formal Description of System/360*, *IBM Systems Journal*, 3(2), 198—261 (1964).
8. Lincoln N. R., *It's Really Not as Much Fun Building a Supercomputer as it Is Simply Inventing One*, in D. J. Kuck et al., eds., *High Speed Computer and Algorithm Organization*, New York, Academic, 1977, pp. 1—11.

---

## ОТВЕТЫ К УПРАЖНЕНИЯМ

**1.1.** И да и нет. Да, поскольку большинство существующих систем разрабатывались именно в такой последовательности, хотя полученные в конечном счете результаты были, как правило, далеки от оптимальных. Если бы вопрос начинался со слов «Обязательно ли», то ответ должен быть отрицательным, поскольку трудно сформулировать требования, установить критерии и принять компромиссные решения, когда архитектура системы еще не определена.

**1.2.** В идеале это должен быть человек, обладающий знаниями всех перечисленных специалистов. Если необходимо сделать выбор, то, вероятно, разработчик компиляторов является наиболее подходящей кандидатурой. Программист, по всей видимости, мало знаком с архитектурой ЭВМ. Разработчик операционной системы склонен оказывать отрицательное влияние на выбор архитектуры, поскольку, хотя результаты разработки операционной системы и являются важными, конечная цель проектирования большинства систем — обеспечение эффективности разработки и выполнения прикладных программ пользователей. Инженер, проектирующий центральный процессор, вероятно, самый плохой кандидат, если только не имеет достаточного представления об интерфейсе аппаратных средств с программным обеспечением, т. е. не знаком с программным обеспечением.

**1.4.** Для Системы 370 фирмы IBM  $S=240$  и  $M=768$ , для микропроцессора 8080 фирмы Intel  $S=192$  и  $M=488$ . Заметим, что здесь должны быть приняты определенные допущения. Так, должно быть определено, как производится выборка команды: 1) команда выбирается полностью; 2) выбирается содержимое только тех полей команды, которые необходимы для выполнения операции (например, если условный переход не должен выполняться, то процессору не нужно выбирать содержимое поля адреса перехода команды). Приведенные ранее значения параметров  $S$  и  $M$  вычислены с учетом первого предположения.

**1.5.** В системе, построенной на базе архитектуры семейства микропроцессоров 8080 фирмы Intel, объем памяти, требуемый для выполнения программ, составляет приблизительно 80% объема памяти в Системе 370, причем программы выполняются на 35% быстрее. Разумеется, необходимо проанализировать большое число примеров программ, прежде чем сделать это заключение.

**1.6. 8,3%.**

**1.7.** Основная причина — использование микропроцессорами 8080 фирмы Intel 8-битовых команд, выполняющих приращение содержимого регистра или пары регистров на 1 (наиболее часто встречающаяся операция). В Системе 370 для той же цели требуется выполнение 32-битовой команды.

**1.8.** Нет. Убедительным подтверждением этого является организация пересылки нескольких значений данных из одной области памяти в другую (часто используемая операция). В Системе 370 такую операцию выполняет

одна команда, а в микропроцессорах 8080 фирмы Intel для этого должен быть составлен цикл из нескольких команд.

**1.9.** Как и странно, но критерий стоимости. (Странно, потому что производителем ЭВМ является промышленность.) Архитектурные решения влияют на стоимость процессора, требуемый объем памяти, стоимость разработки компиляторов, системного программного обеспечения (в том числе операционных систем) и прикладных программ. Перечисленные затраты связаны между собой достаточно сложным образом, причем соотношения до сих пор не были широко исследованы.

**2.1. 1.** Массив может быть многомерным, но память машины фон Неймана является линейной, и, следовательно, компилятор должен преобразовывать массив в вектор.

**2.** Размеры элементов в разных массивах могут быть различными, однако аппаратные средства вычислительной системы обычно предоставляют только механизм индексирования, ориентированный на доступ к слову или байту.

**3.** Возможно выполнение операций сразу над всем массивом, однако модель памяти машины фон Неймана предполагает единовременную обработку только слова.

**4.** Кросс-секции предусматривают обращения к группам данных, расположенных в различных, несмежных участках памяти; однако модель памяти фон Неймана этого не допускает.

**5.** Согласно принципам архитектуры машины фон Неймана, модель памяти — это одни большой вектор, охватывающий всю запоминающую среду машины; при этом отсутствуют какие-либо явно обозначенные границы местоположения совокупностей данных. Все это указывает на неявный (для модели памяти машины) характер описания таких моделей данных или структур, как массивы: причем эти описания даются в программе.

**2.2.** Попытки слишком сильного сокращения семантического разрыва могут на практике привести к обратному эффекту. Предположим, например, что при разработке архитектуры машины удалось свести почти к нулю ее семантический разрыв с языком ПЛ/1. Это может оказаться эффективным для программирования на языке, но не исключено (и даже весьма вероятно), что при этом увеличится семантический разрыв между архитектурой и другими языками (такими, как Ада, Паскаль, КОБОЛ) по сравнению с разрывом между языками и архитектурой традиционных машин. Поэтому одна из возможных стратегий при решении подобной проблемы для «многоязычной» вычислительной системы — сокращение разрыва между ее архитектурой и семантическими характеристиками, являющимися общими для языков программирования. Существуют и другие стратегии решения этой проблемы, но они рассматриваются в четвертой и пятой частях книги.

**2.3.** Для сокращения семантического разрыва можно рекомендовать следующее:

1) вместо памяти в виде единого последовательного адресного пространства использовать запоминающую среду, которая бы более походила на модель памяти языков программирования;

2) память должна допускать представление ее модели как многомерного пространства;

3) машина должна отличать данные от программ по форме их представления в памяти;

4) характеристики данных следует хранить вместе с данными, вместо того чтобы включать их в качестве описаний в программы.

**2.4.** Если для представления чисел используется двоичная система счисления, то двоичные эквиваленты десятичных чисел являются только их приближенным представлением (например, десятичной дроби 0,1 эквивалентна бесконечная дробь 0.000110011001..., подлежащая усечению).

**2.5. 20,4%.** Отношение количества двоичных разрядов, занимаемых десятично кодированными числами, к количеству разрядов их двоичных эквива-

лентов равно  $4 \log 2$  (в предположении, что равновероятны все значения чисел).

**2.6.** Такое решение позволит достичь немногого, поскольку частота вызова процедуры (требующего сохранения содержимого регистров с их последующим восстановлением) значительно больше, чем частота переключения процессов или прерываний.

**2.7.** Указанное отношение равно 132/227, причем 227 регистров были загружены в процессе выполнения программы, но только содержимое 132 из них было использовано. Если же учесть, что команды BALR, LTR и CVB тоже загружают регистры, то упомянутое отношение окажется еще меньше.

### 3.1. Возможны следующие подходы:

1) ориентация архитектуры машины на использование только одного языка программирования;

2) при ориентации архитектуры на несколько языков программирования использование только тех семантических характеристик этих языков, которые являются для них общими (т. е. отказ от ориентации архитектуры на один конкретный язык программирования);

3) использование мультипроцессорной конфигурации вычислительной системы, каждый процессор которой связан с конкретным языком высокого уровня, а операционная система выполняет функцию диспетчеризации программ по соответствующим процессорам;

4) оснащение процессора набором микропрограмм, способных реализовать то или иное требуемое архитектурное решение, при условии, что операционная система выполняет функцию планировщика переключений на требуемую микропрограмму (подход, реализованный в ЭВМ B1700 фирмы Burroughs);

5) введение так называемого базового набора команд, используемого всеми языками программирования, и группы дополнительных наборов команд, учитывающий специфические требования каждого из языков (подход, реализованный в машине SWARD; см. часть 5).

**3.2. БЕЙСИК** (широко используется в персональных ЭВМ) и РПГ (применяется в малых ЭВМ для коммерческих расчетов).

**4.1.** Такой может оказаться конструкция языка, в котором «накладываются» друг на друга или объединяются идентификаторы переменных разного типа; именно это осуществляется оператор EQUIVALENCE в ФОРТРАНе, предложение REDEFINES в КОБОЛе и атрибут DEFINED в ПЛ/1. Например, операторы ФОРТРАНА

```
INTEGER A
REAL B
EQUIVALENCE (A,B)
```

назначают переменным А и В одну и ту же область памяти, допуская размещение в ней числа с плавающей точкой при использовании имени В и целого числа при использовании имени А. Однако такая практика программирования признана порочной, поскольку делает программу зависимой от специфики аппаратных средств машины. По указанной причине новые стандарты упомянутых языков программирования, а также описания новых языков (например, языка Ада) запрещают применение подобных средств.

**4.3.** Программа, выполняемая на машине Y, требует меньше памяти, если R больше чем 1,6.

**4.4.** Программа для теговой машины (Y) занимает 85% того объема памяти, который требуется той же программе для машины X. У машины X команды 48-битовые, а у машины Y — 38-битовые. Средний размер операнда машины X составляет 32 бит, а машины Y — 40 бит. Предполагается, что на каждый из указанных операндов в среднем приходится пять команд.

**4.5.** Это пример трудностей, с которыми сталкиваются, когда информация о данных полностью содержится в потоке команд. Так, в Системе 370 команда MVC (ПЕРЕСЫЛКА СИМВОЛА) указывает количество байтов, подле-

жащих пересылке. Часто нельзя заблаговременно «связать» эту команду с конкретным счетчиком байтов (например, желательно задать пересылку формального параметра в виде строки символов, размер которой определяется текущим фактическим параметром). Тогда для хранения этой величины (размера) предоставляется регистр. Подлежащая выполнению команда EXECUTE задает команду MVC как operand. В результате появляется возможность модификации команды MVC посредством содержимого регистра и последующего ее выполнения.

4.6. Появление упомянутых «двоусыленностей» связано с моментом чтения следующей карты. Если имеет место обращение к части этого поля (например, к восьмому символу), предполагает ли это чтение следующей карты? Читается ли карта при выполнении команды сравнения, использующей содержимое поля в качестве операнда? Как представляется и выявляется факт того, что очередь пуста? И, конечно, потребовался бы механизм, предотвращающий недопустимые записи в указанное поле.

#### 4.7.

Адресация посредством

регистров	аккумулятора	стека с безадресными командами	стека с одноадресными командами
LOAD R1,B	LOAD B	PUSH B	PUSH B
ADD R1,C	ADD C	PUSH C	ADD C
LOAD R2,D	STORE TEMP	ADD	PUSH D
ADD R2,E	LOAD D	PUSH D	ADD E
MULT R1,R2	ADD E	PUSH E	MULT
STORE R1,A	MULT TEMP	ADD	STORE A
	STORE A	MULT	
		STORE A	

Адресация типа память — память

с двухадресными командами	с трехадресными командами	с 2/3-адресными командами
MOVE A,B	ADD A,B,C	ADD A,B,C
ADD A,C	ADD TEMP,D,E	ADD TEMP,D,E
MOVE TEMP,D	MULT A,A,TEMP	MULT A,TEMP
ADD TEMP,E		
MULT A,TEMP		

5.1. Содержимое поля типа определяет тип данных элементов массива.

5.2. В память сегментов программ и строк данных, память таблицы символьических имен и память таблицы областей действия переменных.

5.3. Содержимое памяти таблицы символьических имен и памяти таблицы областей действия переменных.

5.4. Указатель массива в стековой памяти данных адресует дескриптор, находящийся в памяти дескрипторов массива. Последний указывает местоположение первого элемента массива в верхней части стековой памяти данных. Элементы массива в стековой памяти данных представлены дескрипторами, адресующими строки символов, записанные в памяти сегментов программ и строк данных.

5.5. Максимальное число равно 16, поскольку в машине имеется 16 дисплей-регистров.

5.6. Изменяется содержимое стековой памяти данных и регистра — указателя стековой памяти данных (при распределении локальной памяти для вызываемой процедуры), памяти сегментов программ и строк символов (если вызываемая процедура использует символьные переменные), памяти стека указателей сегментов программы и стека указателей активизируемых сегментов программы, а также регистров-указателей этих стеков и дисплеев-регистров.

5.7. Не всегда. Каждый элемент должен содержать ссылки на предшествующий элемент стека, за исключением случаев рекурсивных вызовов процедур.

6.1. DS(103) — указатель массива, DS(102) — число 0, DS(101) — число 1. [Ложжение, DS (data stack) — стек данных.]

6.2. Эти команды предназначены для инициализации первого элемента массива (с индексом 0) как дескриптора сегмента программы с адресом 700 (сегмента, соответствующего ситуации ELSE).

6.3. Дескриптор сегмента программы, находящийся в стеке, указывает местоположение вызываемого сегмента. Первой командой сегмента должна быть команда SCOPEID, указывающая «вход» таблицы областей действия переменных, относящийся к данному сегменту программы.

6.4. Количество фактических параметров указывается в поле операндов команд. Количество формальных параметров процедуры определено в соответствующем элементе таблицы областей действия переменных.

6.5. Первая команда удаляет из стека неопределенную величину, возвращаемую после выполнения процедуры ZZZ, вторая — дескриптор косвенного адреса фактического параметра, помещенный в стек при выполнении команды с номером 34.

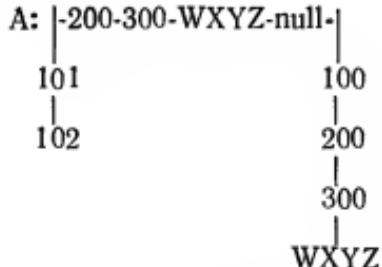
6.6. Элемент таблицы областей действия переменных определяет количество формальных параметров, адрес первого элемента таблицы симвлических имен и количество этих элементов, представляющих параметры. Формальные параметры всегда описываются первыми в таблице симвлических имен, поскольку их имена первыми встречаются в процессе компиляции процедуры (они перечисляются в операторе PROCEDURE).

8.1. Очевидный недостаток — «дороговизна» внесения изменений в систему. Например, при добавлении в набор команд языка SPL нового оператора необходимо изменение конструкции части последовательностных логических схем транслятора.

8.2. Производится обращение к четвертому элементу трехмерной подструктуры, являющейся третьим элементом двумерной подструктуры, в свою очередь являющейся вторым элементом одномерной подструктуры X.

8.3. Возможны оба варианта: результат может быть скаляром или структурой.

#### 8.4.



9.1. Внутренняя форма представления числовых данных — упакованные десятичные числа. Восьмибитовое поле используется для представления не одной цифры, а двух.

**9.2. FD XX XX XX XX XX XX XX**

F5 30 30 30 30 30 30 30

F6 XX XX XX XX XX XX F6

F5 31 31 31 31 31 31

F6 XX XX XX XX XX F6

FF XX XX XX XX XX XX XX

**9.3. F5 31 F6 XX XX XX XX F6**

**9.4. 80 YY YY YY XX XX XX XX**, где YY YY YY — адрес области памяти, выделенной динамически для размещения числа 1234,5678.

**9.5. Из управляющего слова идентификатора с адресом 2222. Содержимое управляющего слова идентификатора с адресом 2272.**

**10.1. Информация, идентифицирующая терминалы и пользователей, передается контроллеру памяти с целью определения последних списков страниц пользователя, к которым относится данный запрос (например, запрос на выделение группы слов).**

**10.2. Центральный процессор использует в качестве стека строку логической памяти. При этом операция контроллера памяти FR (см. табл. 10.2) соответствует удалению данных из стека (операция POP центрального процессора).**

**10.3. Если бы не было поля IGAC, то контроллер памяти должен был бы соединять все слова связи групп слов страницы между собой для образования списка доступных для распределения групп при каждом использовании свободной страницы. Наличие поля IGAC позволяет контроллеру памяти достичь той же цели просто путем записи 1 в поле IGAC перед началом использования свободной страницы. Если страница используется длительное время, в течение которого ее группы распределяются и освобождаются, то содержимое поля IGAC в конечном счете будет увеличено до 29; после этого группы будут распределяться из списка доступных для распределения групп. Таким образом, наличие поля IGAC уменьшает время, затрачиваемое на распределение памяти страниц.**

**10.4. Для эффективного выполнения операции FR (см. табл. 10.2).**

**10.5. Наличие этих операций объясняется потребностью контроллера памяти и регенератора памяти манипулировать одними и теми же списками. Если регенератор памяти удаляет элементы из списка областей памяти с «информационным мусором» или включает их в список областей памяти, доступных для распределения, и при выполнении этого контроллер памяти обращается к указанным спискам для выполнения запросов другого процессора, то может возникнуть ошибка, связанная с несогласованностью действий во времени регенератора памяти и контроллера памяти (в частности, область памяти может быть «утеряна» или включена сразу в несколько списков). Таким образом, операции DL и RG выполняются контроллером памяти для обеспечения последовательной работы со списками областей памяти с «информационным мусором» и доступных для распределения.**

**10.6. Нет. Такая ситуация возникает только при работе со строками, представляющими структуры языка SPL; память для этих строк всегда распределяется из списка TPL2. Следовательно, регенератор памяти должен выполнять подобную процедуру только в случае, когда страница, содержащая строку, включена в список TPL2. Процессор адресации данных также помогает регенератору памяти в выполнении данной процедуры путем присвоения единичного значения биту 37 слова связи групп (табл. 10.3), если в одно из слов помещается адрес.**

**10.7. Адреса и данные сами определяют свой тип. Поля данных начинаются с управляющих символов F0, F1, F2, F3 или Fб, указатели подгруппы структуры — с управляющего символа EC.**

10.8. В системе имеется один общий список свободных (доступных для распределения) страниц; он содержит все ядераспределенные страницы памяти. Для каждого пользователя предусмотрены три списка свободных областей памяти; в них включены страницы, содержащие группы слов, доступные для распределения. На каждой странице имеется один список свободных групп слов; в него входят все свободные, освобожденные регенератором памяти группы слов этой страницы.

10.9. Элемент структуры A[2,2] содержит в данный момент скаляр 202. Для размещения вектора <1234|4321> будет выделена память строки, состоящей из трех слов, и указатель подгруппы структуры будет помещен в слово, содержащее величину 202. Кроме того, второе слово (указатель подгруппы структуры) в крайнем слева блоке на рис. 10.4 будет теперь содержать индекс 02, а его последнее поле будет адресовать указатель этой новой подгруппы.

10.10. Первые 32 слова каждой страницы не могут использоваться в качестве памяти для размещения данных. Они содержат заголовок страницы и слова связи групп слов страницы.

12.1. Значение, равное 6.

12.2. 16 бит (3 — код операции, 7 — поле первого операнда, 6 — поле второго операнда).

12.3. 19 бит.

12.4. Пересылка 4-битового числа без знака 73 в область памяти, относящуюся к дескриптору, индекс которого в таблице дескрипторов равен 27.

12.5. В шестнадцатеричном представлении — число 4EF7F340.

12.6. S\*\*\*\* 10.45 (в коде EBCDIC).

12.7. Наиболее оригинальными атрибутами набора команд, ориентированного на язык КОБОЛ, являются атрибуты команд, выполняющих операции редактирования и сравнения. Они хорошо отражают природу основного применения языка КОБОЛ, поскольку программы на этом языке ориентированы главным образом на ввод-вывод и манипулирование данными<sup>1)</sup> и выполняют относительно небольшой объем арифметических операций.

14.1. Данная ячейка представляет число с плавающей точкой, равное  $-0.7493 \cdot 10^{23}$ .

14.2. Результат работы компилятора — внешний модуль — представлен посредством строки токенов. Определение модуля для машины осуществляется путем выполнения команды CREATE-MODULE, одним из операндов которой является строка токенов, представляющая модуль. Стока (поле) токенов может быть также использована для эмуляции других архитектур; при этом адресное пространство машины другой архитектуры содержится в поле токенов. Дополнительный набор команд может содержать команду «Эмуляция архитектуры X».

14.3. Массивы я записей не могут быть адресатами ячеек «параметр» типа D или «косвенный доступ к данным» из-за неопределенностей («двусмысленностей»), которые при этом могли бы возникнуть в потоке команд (различными являются адреса операндов для скаляров, массивов и записей). Ячейки типа D и ячейки, тип которых определяется пользователем, нельзя употреблять произвольным образом («смешивая» их), поскольку их назначение различно. Использование ячеек типа D означает определение тех или иных атрибутов динамически (в зависимости от конкретных «внешних» условий). Введение же пользователем своих типов (для каких-либо ячеек) объясняется их стремлением снабдить машину дополнительными средствами контроля типа обрабатываемой я информация.

14.4. 1009000F008006A.

14.5. 100900010000090002000 (адреса операндов — это адреса записей).

<sup>1)</sup> Имеются в виду операции пересылки, преобразования и редактирования данных. — Прим. перев.

14.6. Для присвоения начальных значений элементам массива в динамической части адресного пространства компилятор должен сгенерировать одну или несколько команд у каждой точки входа в модуль. Иначе неоднозначно решается задача присвоения начальных значений элементам массива в статической части адресного пространства. Одним из методов является представление к компилятору следующих требований: 1) предоставить модулю дополнительную точку входа; 2) сгенерировать код для присвоения начальных значений имению у этой точки входа. После создания такого модуля подобную точку входа можно было бы назвать «редактор связей».

14.7. Глобальная переменная представляется как ячейка «косвенный доступ к данным» с соответствующей ячейкой «указатель». Редактор связей будет инициировать содержимое ячейки «указатель» посредством команды LINK. При использовании области общей памяти языка ФОРТРАН или структуры с внешним атрибутом языка ПЛ/И содержимым ячейки «косвенный доступ к данным» является запись.

14.8. Такая ситуация может возникнуть, например, при указании в программе, что значение параметра итераций цикла DO не определено при его окончании. Другим примером такой ситуации является присвоение в программе той или иной переменной иулевого значения (отсутствия значения), как это делается в языке ПЛ/И или Ада.

#### 14.9.

Смещение		Комментарий	
01	0001E0004C0009F00121	Header indices	
15	330000000	CAS/IAS/SIS/Faults	
1E 001	F800000	I	
	008 6005F0000000	Q	
	014 6008B0000000000	M	
	023 6005F0000000	N	
4C 02F	70101000600000AF000000	QUASI	
	045 70131001000000AB0083000000	MICH	
	05E 900000000000000000000000000000	BUFFALO	
	074 F00000A	SC10	
	07B F800000	STAA	
9F 001	C03008014023	ACT 3,Q,M,N	
	702F0230000046	EQBF QUASI(N),0,%A	
	102F023008	MOVE QUASI(N),Q	
	0807B014	LENGTH STAA,M	
	0407B0450230080140001	MOVESS STAA,MICH(N),Q,M,1	
	E053	B %B	
046	D05E013045023	%A: CALL BUFFALO,1,RW.MICH(N)	
053	0A	%B: RETURN	
055	C01008	ACT 1,Q	
	10010001	MOVE I,I	
063	202F001001	%C: ADD QUASI(I),I	
	202F001008	ADD QUASI(I),Q	
	5001074063	ITERATE I,SC10,%C	
	0A	RETURN	

14.10. Цикл DO во второй процедуре можно было бы заменить оператором QUASI=QUASI+I+Q; это привело бы к замене команд ITERATE и MOVE, а также двух команд ADD двумя командами сложения, добавляющими значения скаляров I и Q в массив QUASI. Такую возможность мог бы предоставить хороший оптимизирующий компилятор.

17.1. Поскольку объекты доступны программам, естественно желание раз-

решить определенным программам оперировать объектами как абстрактными данными, исключая возможность анализа или модификации представления содержимого объектов. Например, может появиться желание дать процессу дескриптор доступа к порту, разрешая тем самым посыпать этому порту сообщения без права анализа или реорганизации очереди запросов к последнему. Обычно на практике только программа управления типами для объекта (например, операционная система, если речь идет о системных объектах) получает дескрипторы доступа с правами доступа для объектов.

#### 17.2. Сегменты данных.

17.3. Поскольку структура данных портов та же самая, команду SEND можно выполнять как для порта диспетчеризации, так и для порта связи. Попытка процесса в порт диспетчеризации — это один из способов пуска процесса.

17.4. Потому что каждому процессу предоставляется только одно средство переноса данных (один транспортер).

17.5. Функция этих объектов схожа (например, объект «управление дескриптором» расширяет права доступа в дескрипторе доступа с целью выполнения операций над объектом «от имени» источника обращения к объекту). Однако объект «управление дескриптором» определяет такой объект, манипулирование которым возможно только как системным объектом. Поэтому объект «управление дескриптором» не может отличить сегмент данных расширенного типа X от сегмента данных типа Y (оба воспринимаются им как сегменты данных общего типа). Итак, при использовании объектов «управление дескриптором» и средств расширения прав доступа, программа управления типами может быть дезориентирована и направлена на выполнение операций над объектом, тип которого не соответствует этим операциям.

17.6. Если используется глобальный объект «ресурсы памяти», то таковой является таблица объектов, дескриптор доступа которой находится в сегменте доступа объекта «ресурсы памяти». При использовании объекта «ресурсы памяти», локального по отношению к процессу, таковой является таблица объектов, дескриптор доступа которой находится в сегменте доступа процесса.

17.7. Прежде всего это делается из соображений защиты от несанкционированного доступа. Бит младшего разряда в дескрипторе доступа выполняет роль индикатора такой защиты. Если в новых сегментах доступа ему не присвоено нулевое значение, появляется возможность изменения (что недопустимо) потенциальных адресов. Кроме того, присвоение нулевых значений указанным битам позволяет машине в дальнейшем обнаруживать попытки использования неопределенных дескрипторов доступа (которые являются скорее ошибками программирования, а не попытками нарушения установленных правил санкционированного доступа).

17.8. Такая ситуация возникает при желании предоставить процессу возможность обрабатывать ошибки, возникающие при его протекании (т. е. без направления процесса в порт обработки ошибок). Очевидно, что такую возможность желательно предоставить процессу операционной системы, принимающему процессы из порта обработки ошибок.

17.9. Такие ситуации возникают, когда программа обработки ошибок должна выполнять операции, санкционируемые подпрограммой, содержащей ошибку с целью разрешения возникшей особой ситуации (например, переполнения); в подобных случаях программа обработки ошибок должна сама уточнить специфические характеристики контекста с ошибкой, определяющие параметры подлежащих выполнению операций (например, тип округления результата арифметической операции).

18.1. Это объясняется двумя причинами. Во-первых, согласно определению данных вещественного типа, возможно групповое представление одной и той же величины (например,  $+0/-0$ , неnormalизованные величины). Во-вторых, существует возможность представления данных как «не число», что может быть использовано средствами программного обеспечения для обозначения переменной, не получившей начального значения.

18.2. Это команда EQUAL-ORDINAL. Ее первый операнд расположен со смещением 9A в сегменте данных «константы», связанном с контекстом. Второй операнд имеет форму A(I), где A — вектор порядковых чисел, начало которого совпадает с началом сегмента данных, местоположение которого определяется дескриптором доступа, расположенным в точке с индексом 3F в сегменте доступа к элементу 3. I — это короткое порядковое число (как и все индексы векторов) в сегменте данных «контекст» со смещением 1E. Третий операнд — результат сравнения — расположен на вершине стека.

Отдельные части кода команды (анализ начинается справа) имеют следующее назначение:

1010	Класс: три операнда длиной 32, 32 и 8
1100	Формат: операнды 1 и 2 — ссылки 1 и 2; операнд 3 — вершина стека
00	Обращение к скаляру (ссылка 1)
00	Селектор сегмента прямой короткий
1	Длина смещения (16)
00	Использование ВСД 0
0001	Индекс ВСД (для дескриптора доступа для сегмента данных «константы»)
0000000010011010	Смещение (9A)
10	Обращение к элементу вектора (ссылка 2)
01	Селектор сегмента прямой длинный
0	Длина базы (0 — отсутствие базы указывает на начало сегмента)
11	Индекс ВСД (3F)
00000000111111	Индекс ВСД 3 (3F)
00	Индекс косвенный общего назначения
0	Селектор прямой короткий
0	Длина смещения (7)
00	Использование ВСД 0
0000	Индекс ВСД (к сегменту данных «контекст»)
0001110	Смещение (1E)
000	Для класса 1010 означает EQUAL ORDINAL

20.1. Программа имеет следующий вид:

```

MAX [ITEM(VOL)] [REG1]
SELECT MARK (A) [ITEM: ITEM.VOL=REG1]
CROSS-SELECT MARK(A) [EMP: EMP.DEPT = ITEM.DEPT]
[ITEM.MKED(A)]
ADD [EMP(SAL): (EMP.MKED(A) ) AND (EMP.SAL<20000)]
[100]
SELECT RESET(A) [EMP]
SELECT RESET(A) [ITEM]
EOQ

```

20.2. В худшем случае ~350 мс. В среднем задержка по времени, связанная с отысканием начала памяти ячеек с целью пуска программы, составит 10 мс. Для команды MAX требуется несколько больше, чем один оборот вращающегося носителя, но команда SELECT не может начать выполнение до тех пор, пока снова не появится начало памяти. Вот почему вклад команды MAX в общую задержку составляет 40 мс. Каждая из трех команд — одной команды ADD и двух команд SELECT — требует 20 мс. Предполагая наихудшую ситуацию, при которой информация о каждом из восьми служащих находится в отдельной ячейке, получаем, что для выполнения команды CROSS-SELECT требуется 220 мс.

20.3. Такими показателями являются следующие: 1) существенное сокращение объема данных, пересылаемых между рассматриваемым устройством и основной памятью (в традиционных системах средства поиска расположены в центральном процессоре); 2) сокращение числа команд, выполняемых в центральном процессоре для программных средств управления данными; 3) отсутствие необходимости в поиске и обновлении трактов доступа (например, индексов).

20.4. «Сложные» операции в табл. 20.1 включают проверку десяти условий. Однако RAP.2 не располагает десятью компараторами. Поэтому соответствующие операции разделяются между несколькими командами, что в свою очередь требует нескольких обращений к данным для завершения работы.

20.5. Одно из возможных решений имеет следующий вид:

SELECT	MARK(A) [LOCATION: LOCATION.CITY = "MIAMI"]
CROSS-SELECT	MARK(A) [ACCOUNTS: ACCOUNTS. DISTCODE= LOCATION.DISTCODE] [LOCATION. MKED(A)]
SAVE	(1) [CHARGES(MINCHARGE): STATUS='Q'] [REG1]
SELECT	RESET(A) [ACCOUNTS: PHONES<=1   STATUS= 'Q'   MCHARGE=REG1]
SUM	[ACCOUNTS(MCHARGE): ACCOUNTS.MKED(A)] [REG1]
READ-REG	[REG1] [MYBUF]
SELECT	RESET(A) [ACCOUNTS]
SELECT	RESET(A) [LOCATION]
EQQ	

20.6. В предположении, что для начала требуется в среднем 0,5 циклических просмотра, время обработки равно 10,5 просмотра; для команды CROSS-SELECT требуется 3 просмотра, а для команды SUM — 2 просмотра.

20.7. После выполнения маркирования элементов отношения можно ожидать, что многие из маркированных элементов принадлежат одному и тому же отделу. В этом случае можно было бы посредством операции проектирования (итерационного использования команд GET-FIRST-MARK, SELECT, RESET и BC) устраниć дублирующие друг друга маркированные отделы прежде, чем применять команду CROSS-SELECT.

20.8. При K=5 и без предварительного выполнения операции проектирования потребуется 43 сканирования ( $1+200/5+2$ ), чтобы выполнить команду

**CROSS-SELECT.** Для выполнения операции проектирования — десять итераций команд **GET-FIRST-MARK**, **SELECT** и **BC** — необходимо 30 сканирований. Для реализации после этого команды **CROSS-SELECT** требуется 5 сканирований ( $1+10/5+2$ ). В результате общее число необходимых сканирований сокращается на 8.

**22.1.** Следует добавить вентили типа Т во входные ветви типа Т трех верхних блоков соединения. Эти вентили получают сигналы управления в результате проверки отношения двух operandов по критерию «меньше или равно».

**22.4.** Величина 42 заменяется структурой из двух величин 3,6 и 7.

**23.1. а) Команда LITERAL:** поместить целое число +36 на вершину стека.

**б) Команда SNAME:** поместить дескриптор косвенного адреса, соответствующий адресу лексического уровня 1,35, на вершину стека.

**в) Команда SNAME:** поместить дескриптор косвенного адреса, соответствующий адресу лексического уровня 1,3, на вершину стека.

**г) Команда SNAME:** поместить дескриптор косвенного адреса, соответствующий адресу лексического уровня 1,3, на вершину стека.

**д) Команда SNAME:** поместить дескриптор косвенного адреса, соответствующий адресу лексического уровня 3,3, на вершину стека.

**е) Команда SLOAD:** поместить на вершину стека величину, хранимую по адресу лексического уровня 1,3.

**23.2.** Значение первых двух битов кодов операций других команд должно равняться 11, в противном случае машина не могла бы безошибочно проводить различные между командами.

**23.4.** Вероятно, меньшими, но определенный ответ дать трудно. С одной стороны, в машине более высокого уровня языка программирования команды менее примитивны, и поэтому их последовательности в большей степени зависят от того, как составлена исходная программа (чего нельзя сказать о последовательностях команд, генерируемых компиляторами при реализации той или иной конструкции конкретного языка программирования). Так, например, машина **SWARD** предоставляет немного очевидных возможностей для оптимизации такого типа. С другой стороны, для машин высокого уровня языка программирования характерна тенденция к использованию меньшего количества команд (и следовательно, меньшего количества уникальных попарных сочетаний команд), что повышает частоту использования конкретной пары команд.

**23.5.**  $H=1,79$  бит. Избыточность равна 40%. Если существует зависимость последовательности появления команд, то показатель  $H$  для потока кодов операций становится меньше 1,79 бит.

**23.6.** Вместо того чтобы в качестве кода переключения использовать только комбинацию битов 0000, введем в качестве другого возможного значения этого кода комбинацию 1111. Например, пусть XXXX обозначает комбинацию из 4 бит, для которой недопустимы являются сочетания 0000 и 1111. Тогда 4-битовые коды операций представляются в виде XXXX, 8-битовые коды — как 0000XXXX или 1111XXXX, 12-битовые коды имеют вид 00000000XXXX, 00001111XXXX, 11110000XXXX или 11111111XXXX.

**23.7.** Указанный способ кодирования дает приблизительно такой же результат, как и методы 15/15/... и 15/15/256. Средняя длина кода операции равна 4,87 бит.

**23.8.**  $H=8$  при условии, что появление всех команд равновероятно и при этом имеется именно 256 различных друг от друга кодов операций (маловероятная ситуация).

**23.9.** Может быть определено 10 4-битовых кодов операций, 20 6-битовых кодов и 64 10-битовых кода (или 63 10-битовых кода, если одна комбинация 10 бит резервируется в качестве кода переключения для использования дополнительных кодов операций).

**23.10.** Изучение статистических характеристик языков программирования

и анализ наборов команд других вычислительных систем показывают, что большинство команд передачи управления содержат ссылки на адреса с более высокими номерами (чем адрес самой команды передачи управления). Следовательно, использование относительного смещения со знаком можно считать неэффективным решением. В качестве альтернативы можно предложить расширение диапазона значений смещения без знака «вперед» (в сторону больших значений адреса), используя самые большие значения адресов для редко встречающихся обращений «назад» (к адресам с меньшими номерами). Согласно той же самой статистике, упомянутой выше, при выполнении команды передачи управления перемещение управления «вперед» относительно мало и обычно не превышает длины машинных кодов одного или двух операторов языка высокого уровня и, следовательно, значительно меньше 512-битового диапазона значений относительного смещения. Поэтому, например, замена 10-битового относительного смещения со знаком на 8-битовое относительное смещение без знака (положительное смещение) может дать ощутимый выигрыш.

**23.11.** При таких обращениях к данным при использовании наиболее компактных форм представления (в большинстве случаев в виде прямых коротких скаляров) требуется 18 бит для каждой ссылки. Подобное представление следует признать существенно избыточным. В качестве альтернативного решения можно предложить использование нового типа ссылки к данным, имеющего вид

локаль\_скаляр\_ссылка: := смеш7 1,

где ссылка на сегмент данных «контекст» задается в явном виде (значение неявно заданного селектора сегмента равно 0000 00). Для этого вместо 18 бит достаточно 8 бит. С целью дальнейшего сокращения избыточности параметр смеш7 можно было бы определить как байтовое смещение относительно конца данных (определяемых системой) в сегменте данных «контекст». (Для того чтобы гарантировать корректную идентификацию ссылок к данным такого типа, следует ввести изменения в форму представления трех существующих типов (см. рис. 18.2), а именно использовать для их обозначения не 2 бит, а 3 бит.)

**23.12.** Ответ утвердительный. Следует ввести команду итерации.

## ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ<sup>1)</sup>

Адресация значениями 104  
— косвенная 25, 60  
— лексическая многоуровневая 159, 172  
— потенциальная 110, 111, 115—123, 300, 306, 312, 345, 64, 72—76, 92—95  
— косвенная 119, 306, 320

Ассоциативность частичная 167—171, 207—209

Аффинаж 76, 79

База данных 163—172  
— — реляционная 173—174

Блокирование объектов 111—113

Вектор фиктивный 95

Вызов процедуры 101, 103—110, 230—232

Границы динамические 323

Данные абстрактные 99

Дескриптор 91, 93—99, 270—273

Дисплей-регистр 160, 172

Домен 174

Запись активации 110, 330

Защита от несанкционированного доступа 100, 102

Индексы 165, 193, 209

Команды, наварантные к типу обрабатываемых данных 80, 86, 97, 305

Контроллер каналов ввода-вывода 206  
— памяти на дисках 207

Кортеж 174

Кэш-память 98, 110, 266, 56, 220

Локальность ссылок 192, 278

Машинка DBC 204—216  
— — защита от несанкционированного доступа 211  
— — набор команд 212  
— — представление данных 211  
— — структура системы 205, 209—211

Машинны, управляемые потоком данных 218—253  
— — — обработка структур данных 249—253  
— — — передача пакетов 240—245  
— — — принцип действия 221—223  
— — — тупиковые ситуации 245—249  
— — — язык программирования 223—232

Массивы 37, 93—98, 342—344, 269, 313

Обеспечение программное 12, 257—259  
— — надежность 35—38, 290—299  
— — стоимость 67—70, 126—127, 290—292, 58—59

<sup>1)</sup> Номера страниц, набранные прямым шрифтом, относятся к книге 1, набранные курсивом — к книге 2.

Область санкционированного доступа 100—103, 302—303, 65  
 Обнаружение ошибок 292, 297  
 Обработка мультипроцессорная 27,  
     44, 204—210, 259, 219  
     — ошибок 67, 119—124  
 Объект 99, 111—121, 60—64, 76—84  
 Операция проектирования 175, 183  
     — соединения 174, 182  
 Оптимизация представления адресов  
     275—280  
     — системы команд 258—266  
 Отношение 174  
 Ошибки семантические 292—299

Память ассоциативная 163—172  
     — виртуальная 227, 239—246  
     — на ЦМД (цилиндрических магнитных доменах) 127, 170  
     — одноуровневая 123—130, 301—303  
     — теговая 80—93  
 Параметр АМ 140, 32—34, 45—47,  
     92  
 Параметры M, R и S 19—22, 55,  
     154—156  
 ПЛ-машина учебная 165—202  
 Преобразование адресов 159—162  
 Прерывание 26  
     — «неопределенное» 25  
 Программа-отладчик 258  
 Процессор интерфейсный 62, 69—71  
     — обработки сбоев 207  
     — центральный 251—256  
     — CASSM 196—204  
     — GDP 61  
     — RAP 173—195

Разрыв семантический 27—45  
 Регенератор памяти 207  
 Регистры 52, 55

«Сбор мусора» 121, 242—244, 116—119, 178  
 Сегменты 72—78, 80, 81  
 Сеть переключающая 234  
     — распределительная 234, 241  
     — селекторная 234, 241  
     — трактов передачи пакетов 240—245  
     — управляющая 234, 241  
 Синхронизация процессов 137, 108—113  
 Система В1700 263—267  
     — iAPX 432, 72—127  
     — SWARD 290—363  
     — SYMBOL 219—261  
 Спецификатор типов объектов 78, 79  
 Срез 95  
 Стек активации 104—108, 158—161  
 Супервизор системы 207, 246—251

Типы адресации 144—150  
     — данных 72—76  
 Транспортабельность объектов 130

Указатель доступа (индекс) 165  
 Управление подпрограммами 65  
     — процессами 134—144

Хип-флаг 91

Шина 235, 236

ОН-блок 30, 214, 223

---

## ОГЛАВЛЕНИЕ

Глава 15. Набор команд системы SWARD . . . . .	5
<b>Часть VI. Микропроцессор с архитектурой, ориентированной на объекты.</b> . . . . .	58
Глава 16. Основные принципы работы микропроцессора iAPX 432 . . . . .	58
Глава 17. Процессор общего назначения iAPX 432 . . . . .	72
Глава 18. Набор команд процессора общего назначения . . . . .	128
<b>Часть VII. Архитектура базы данных . . . . .</b>	163
Глава 19. Системы с ассоциативной памятью . . . . .	163
Глава 20. Реляционный ассоциативный процессор . . . . .	173
Глава 21. Другие машины базы данных . . . . .	196
Глава 22. Архитектура машин, управляемых потоком данных . . . . .	218
<b>Часть VIII. Вопросы, связанные с архитектурой систем . . . . .</b>	256
Глава 23. Оптимизация и настройка архитектуры вычислительной системы . . . . .	256
Глава 24. Практические рекомендации по проектированию архитектуры ЭВМ . . . . .	284
<b>Ответы к упражнениям . . . . .</b>	294

**Уважаемый читатель!**

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присыпать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., д. 2, изд-во «Мир».

**ГЛЕНФОРД ДЖ. МАЙЕРС**

## **АРХИТЕКТУРА СОВРЕМЕННЫХ ЭВМ**

Научный редактор Л. А. Паршина  
Младший научный редактор М. Ю. Григоренко  
Художник В. В. Дунько  
Художественный редактор Н. М. Иванов  
Технический редактор И. И. Володина  
Корректор Н. А. Гирия

ИБ № 4069

Сдано в набор 05.10.84. Подписано к печати 18.02.85. Формат 60×90 $\frac{1}{4}$ е. Бумага кн.-журн. № 2. Печать высокая. Гарнитура литературная. Объем 9,75 бум. л. Усл. печ. л. 19,50. Усл. кр.-отт. 19,50. Уч.-изд. л. 20,25. Изд. № 20/3333. Тираж 30 000 экз. Зак. 429. Цена 1 р. 80 к.

ИЗДАТЕЛЬСТВО «МИР»  
129820, ГСП, Москва, И-110, 1-й Рижский пер., 2.  
Московская типография № 11 Союзполиграфпрома  
при Государственном комитете СССР по делам  
издательства, полиграфии и книжной торговли.  
Москва, 113105, Нагатинская ул., д. 1.

Издательство «Мир»  
выпустило в 1984 году книгу

**Питерсон Дж. Теория сетей Петри и моделирование систем:** Пер. с англ.— М.: Мир, 16,50 л., 1 р. 70 к.

Книга американского ученого является первой переводной монографией, посвященной сетям Петри. В ней последовательно излагаются основные понятия теории сетей Петри, задачи, связанные с сетями Петри, методы их анализа. На протяжении всей книги внимание читателя акцентируется на прикладных аспектах теории сетей Петри.

Книга написана просто и доходчиво, все излагаемые понятия подробно обсуждаются. В ней содержится большое число примеров, иллюстраций, упражнений и тем для серьезных исследований. Изложение ведется на должном уровне математической строгости.

Предназначена для научных работников, аспирантов и студентов, имеющих отношение к вычислительной технике и системам распределенной обработки информации.

**В 1986 году  
в издательстве «Мир»  
выйдет книга**

**Молекулярные электронные устройства.** Под ред.  
**Ф. Картера.** Пер. с англ.—М.: Мир, 29 л., 3 р. 20 к.

Книга посвящена новейшему направлению электроники — молекулярной электронике. Рассматриваются вопросы прохождения сигналов в молекулярных агрегатах и их переключения, материалы для изготовления молекулярных устройств и соответствующая микротехнология. Обсуждаются возможности построения информационно-логических систем на базе биологических молекул.

Для специалистов, занимающихся созданием электронных устройств и ЭВМ.







